図解 16ビットマイクロコンピュータ

8086の使い方

井出裕巳 著

●オーム社

図解　16ビットマイクロコンピュータ

8086の使い方

井 出 裕 巳 著

オ ー ム 社

はしがき

マイクロコンピュータも誕生以来10年以上を経過し，いまでは民生用，産業用を問わずあらゆる分野にその応用が定着してきている．現在までのところ，4ビットおよび8ビットCPUが数量的にも圧倒的に多く，それぞれの使用分野に適したところで使用されており，今後ともこの傾向は変わらず，ますますこの部分の応用も拡大してゆくものと思われる．

一方NC，ロボット等に代表される先端の産業分野では，処理速度，能力等の点で8ビットCPUでは限界にきていることから，16ビットCPUへの移行が急速に行われている．現在入手可能な16ビットCPUでは，製造プロセスの改善およびアーキテクチャ上の工夫により，従来の8ビットCPUに比較して約1桁の処理能力の向上が期待できる．

このように，16ビットCPUは，8ビットCPUの上位およびミニコンの下位のアプリケーションの一部を侵食するとしても，大勢としては共存して発展してゆくものと思われ，最近急速に脚光をあびてきたOA，FA，LA等のほか，画像処理などの新しい分野の応用が広がるものと思われる．

以上のような背景の下で，本書は現在16ビットCPUの導入を検討されている技術者だけでなく，新しく学習を始められる方々にも読んで頂けるように，できるだけ図表と対比させることによりわかりやすく記述するよう努めた．また，8ビットCPUに慣れている方々には，それと関連づけて理解できるよう配慮した．

また本書は，設計段階における手引書としても使用して頂けるように，各素子のピン配列，タイミングチャートのほか，巻末に付録として命令一覧表，電気的仕様等も載せた．本書が読者諸氏の8086導入に多少なりと役立ってくれれば幸いである．

なお，本書執筆にあたり，出版および図面の引用等の御承諾を頂きました米国インテルコーポレーション，およびインテルジャパン(株)加茂氏，松本氏に深く

感謝致します．また本書の編集，出版に際してはオーム社の方々に多大の御勘力を頂き，この出版にこぎつけられたことを付記するとともに，厚くお礼申し上げます．

昭和57年9月

著者しるす

目　　次

5. プロセッサ動作のコントロール

6. 命令セット

7. アドレッシングモード

8. システムの構成

9. 周辺ファミリチップ

1. 16ビットマイクロコンピュータと8086

マイクロコンピュータが誕生して10年余り経過し，いよいよ第4世代ともいうべき16ビットの時代に突入である．ここでは8086に至るまでの経緯と，その位置づけ，機能上の特徴および今後の発展方向などについて述べている．

1・1 マイクロコンピュータの発展の歴史

世界最初の**マイクロコンピュータは**，1971年インテル社発売の4004である．当時はランダムロジック置換えによるハードウェア製造コストの低減，多品種少量生産時の仕様変更等をソフトウェアで対応するため，ECRや簡単なコントローラ等への応用が主体であった．その後8ビット並列処理CPUへの強い要望から1972年に8008が登場したが，製造プロセスがPMOSだったので処理能力はあまり向上せず，1973年発売の8080に移っていった．

8080はNMOS製造プロセスで，1命令サイクルが2μsとなり，従来のPMOS形に比べて一躍1桁の処理速度の向上を見，現在でも世界の標準品として広く使われている．その後，ロースレッシュホールドのNMOSシリコンゲートによる単一5V電源**8085A**や**Z80**等が1976年末ごろから次々と発売され，現在の8ビットCPUの主流となっている．このころからCPUの開発はアプリケーションの分野ごとに別々の方向へ分化し始め，8085Aとほぼ同時期に1チップCPU **8048**シリーズが登場した．これは従来，発振器，メモリ，I/Oチップ等複数のチップで構成されていたシステムの機能を1チップに集積することにより，1チップで小規模システムの実現を可能にした．その後用途に応じた4ビットCPUも次々と発売されて家電製品等民生機器への組込みも盛んになり，使用数量で比較すると4ビットのものが圧倒的に多い．

一方複雑なコントロールやデータ処理等の分野ではCPU機能の向上が絶えず要求され，現在の8ビットCPU能力の限界から，16/32ビットCPUの要望も強く，ハードウェアの機能としてはミニコンや中/大形コンピュータにせまるアーキテクチャのものまで発表されている．**8086**ファミリは最初の16ビットCPUとして1978年に発表され，8ビットCPUから16ビットへ移行しやすいようアーキテクチャ上の工夫がされており，データを8/16ビットの両方で扱うことができる．今後は，8086のアーキテクチャを基本に，ハードウェア上の改良および**OS***をシリコン上に集積することにより，ソフトウェア開発の負担を軽減する

* オペレーティングシステムの略．システムを効率良く働かせるためのソフトウェア．

方向へと発展する．**iAPX432**（12・3節）のように高級言語まで組み込んだ32ビットCPUの開発も進行している．

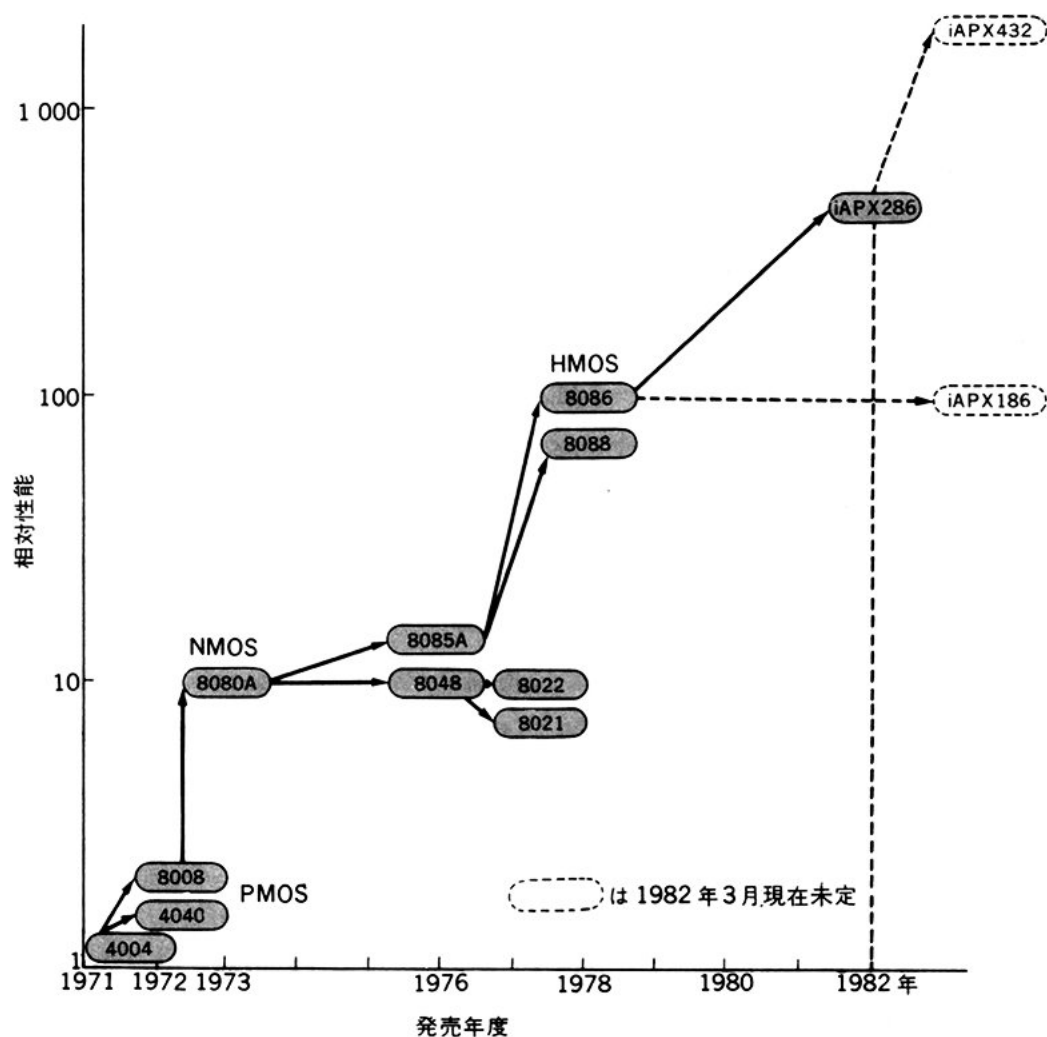



図 1・1　マイクロコンピュータの発展

1・2　8ビットCPUから16ビットCPUへ

8ビットCPUの普及が進むにつれて，マイコンをNC装置，ロボット，画像処理およびデータベースのコンピュータ等と，さらに高度な応用に使用する場合に処理速度，各種演算能力およびアドレッシング機能などで，ミニコンに比較してかなり劣っている点が指摘された．そのため従来の8ビットマイコンのパフォーマンスを1桁以上向上させる必要がある．その解決策には，アーキテクチャ上の改善(16ビット構成の新しいアーキテクチャの採用)，新しい製造プロセス(**HMOS**：ハイパフォーマンスMOS)の採用という両面のアプローチが必要である．

8080等がかなり普及しているので，16ビットへの移行を容易にするため種々の配慮が必要である．すなわち0～1Mバイトのアドレス空間は物理的には16ビットワードの512Kワードとなっているが，論理的には0～1Mバイトのバイトとして扱われ，1ワードのデータは連続した2バイトで構成される．同様にデータやバスも16/8ビット両方の扱いが可能で，従来8ビットCPU用に開発された周辺チップ等もそのまま使用できる（メモリ構成については3・1節参照）．

またレジスタとしては，8086で新たに追加されたセグメントレジスタやインデックスレジスタ（後述）以外の8080と共通のレジスタについてはできるだけ同じ構成とし，抵抗なく移行できるよう考慮している．ただしアキュムレータは16ビットに拡張され，フラグレジスタは下位8ビットは8080と共通であるが，上位バイトは新たに四つのフラグが追加された(2・2節参照)．

割込みのコントロールは80CPUファミリチップである8259Aと組み合わせたベクタ割込みとなっており，基本的な使用法には変わりがない．

ソフトウェアの面では，アセンブラおよび機械語は直接互換性はないが，8080/85のプログラムを8086用に変換するユーティリティ(**CONV-86**)があり，その出力を86のアセンブラ**ASM86**で再アセンブルして，ソースプログラムレベルでの互換性を確保している．コンパイラの場合，**PL/M-86**はPL/M-80と上位互換性があり，そのままPL/M-86でコンパイルできる．

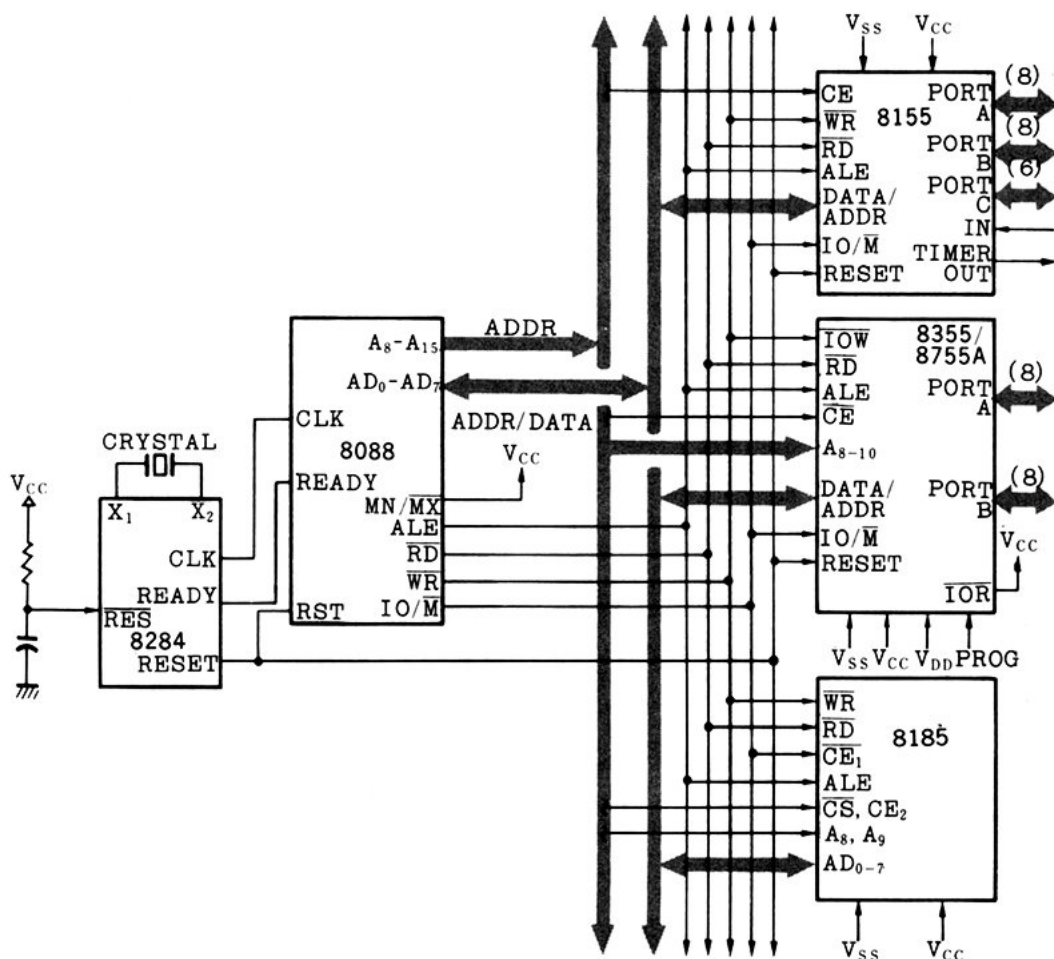



図 1・2 8088による最小システムの構成例

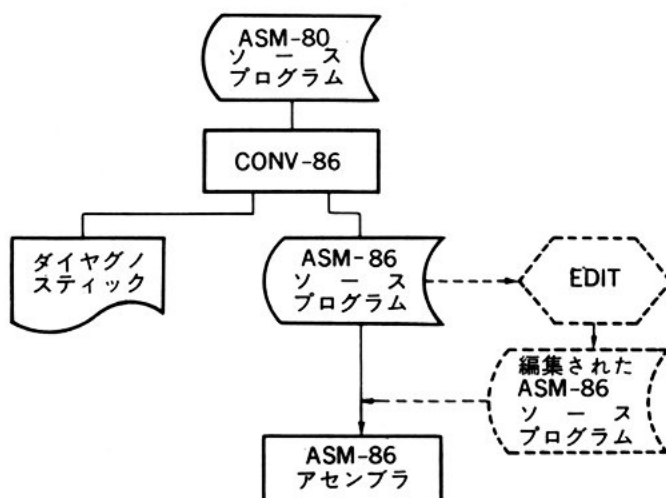



図 1・3 8080プログラムの8086プログラムへの変換

1・3 8086の位置づけと16ビットになって強化された点

8086は最初の16ビットマイクロコンピュータであるということから，これから発展してゆく16ビット以上のCPUの基本アーキテクチャとなるということで重要である．8ビットからの移行をできるだけスムーズに行うという配慮は，かえって16ビットCPUとしてはすっきりしていない点も残るが，周辺チップを含め8ビットCPUファミリの蓄積が生かされるという利点は大きい．

8086と競合する16ビットCPUとしては，モトローラ社の**M68000**およびザイログ社の**Z8000**があり，チップとして市販される汎用16ビットCPUとしては，ほぼこの3機種にしぼられた観がある．これらの比較については他にゆずり，ここでは16ビットコンピュータとして従来の8ビットに比較して強化された点について考察する．

1．**アドレス可能範囲の拡張**　オフセット（IP）によりアドレスされる64Kバイトを単位として，セグメントレジスタとの組合せで1Mバイトまで可能である．また，これによりプログラムのモジュール化が容易になった．

2．**処理速度の向上**　HMOS製造プロセスの導入により，CPUの処理速度が5～10倍向上した．標準5MHzのものでレジスタ・レジスタ間動作0.6μs，8MHzの高速版では0.37μsである．

3．**演算機能の強化**　8/16ビットの符号付きまたは符号なし演算，10進演算，およびハードウェアによる16ビットの乗除算機能等が追加された．

4．**アドレッシング機能の強化**　①リテラル：8/16ビットの直接データ指定，②セグメントレジスタ相対：D_{16}，③レジスタ間接：(BX)，(SI)，(DI)，(BP)，④レジスタ相対：(BX/SI/DI/BP)＋D_8/D_{16}，⑤ベースレジスタ プラスインデックス：(BX/BP)＋(SI/DI)，⑥インデックス修飾されたベースに対する相対：(BX/BP)＋(SI/DI)＋D_8/D_{16}（7章参照）．

5．**ブロックムーブ/ブロックサーチ機能**　インデックスレジスタSI，DIと，命令の前に付加される1バイトのプリフィックス命令による．

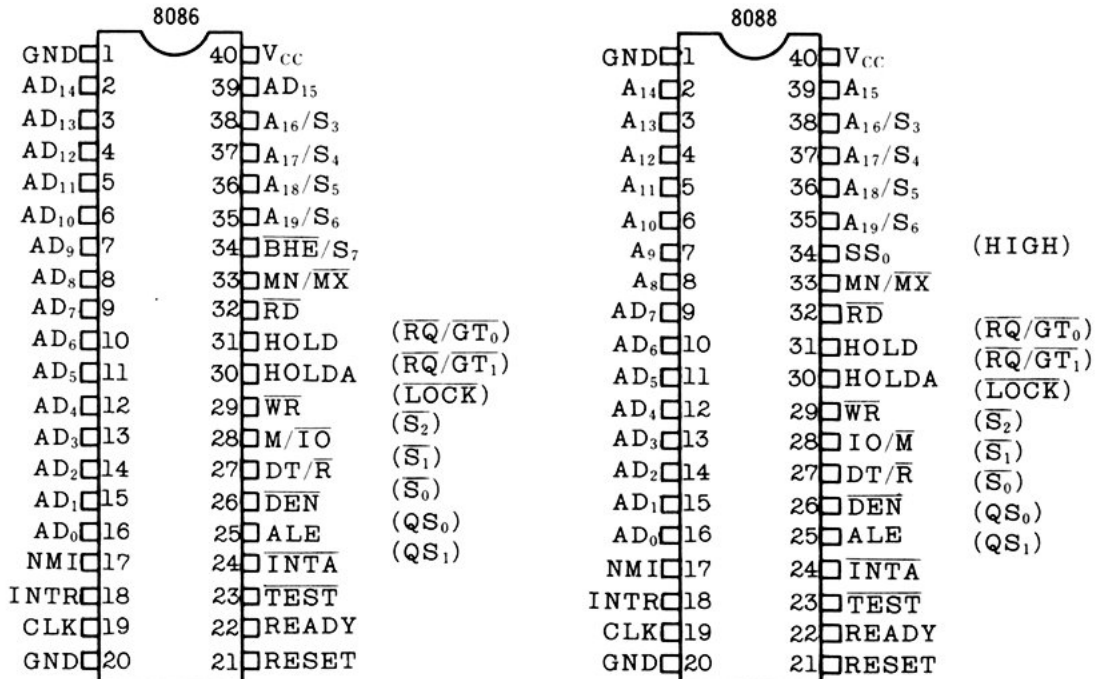
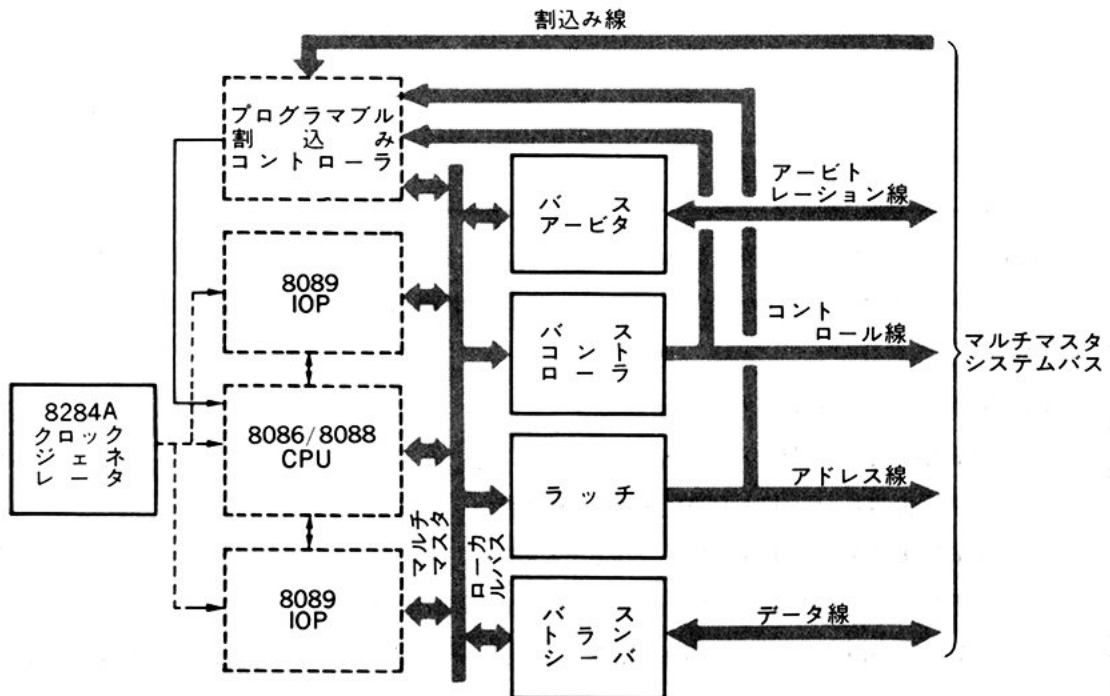
6．**マルチCPUへの対応**　マルチバス互換の周辺ファミリ(8288/89)．

〔注〕 カッコ内はマキシマムモードの場合

図 1・4 8086/8088 のピン構成

図 1・5 基本的なマルチマスタプロセッサシステムの例

8086ファミリでは，CPU本体だけでなく，それと組み合わせて使用されるコ・プロセッサ（8087）やI/Oプロセッサ（8089）が供給され，以下のような呼び名が使用される．

iAPX86（ホストCPUが8086の場合）

iAPX86/10：CPU単独（8086）

iAPX86/11：CPU+IOP（8086+8089）

iAPX86/20：CPU+NDP（8086+8087）

iAPX86/21：CPU+NDP+IOP（8086+8087+8089）

またCPUが8088の場合はiAPX88となり，その組合せによる変形は8086の場合と同じである．

2. 8086のアーキテクチャ

8086の基本アーキテクチャについて述べ，レジスタおよびフラグの構成等を8080/8085と対比して考察し，次に新しく追加されたセグメントレジスタの使用，およびアドレスの生成などについて解説する．また，アドレスバス/データバスの構成についても記述している．

2・1 実行ユニット(EU)とバスインタフェースユニット(BIU)

一般にマイクロコンピュータの基本動作としては，外部メモリやI/Oとのインタフェース部分，およびその読み出された命令やデータに基づき命令を実行する部分より構成されている．8086のアーキテクチャの特徴の一つとして，このメモリ/IOとのインタフェース部（**BIU**）と命令実行部（**EU**）が別々に並行して動作できるよう工夫されており，命令のフェッチと実行が時間的にオーバラップして行われ，アーキテクチャ面から処理速度の向上に役立てている．

バスインタフェースユニットは図2・1の右の部分で，メモリおよびI/Oのアドレスを指定するための20本のアドレスバス（A_0～A_{19}），命令およびデータの入出力のための16本（8088は8本）の双方向性データバス，CPUの内部状態を示す状態情報線（S_0～S_7で一部，アドレスバスと共通），および数本のコントロール線から成っている．前記の並列動作を可能にしているものとして6バイト分（8088は4バイト）の**命令キュー**という考えが導入され，これは**FIFO**（ファーストイン・ファーストアウト）のRAMとしての働きをし，バスインタフェースユニットが実行ユニットの動作とは独立に，6バイトまでの命令を先行して外部メモリからプリフェッチすることを可能にしている．

バスインタフェースユニットは，メモリのアドレスを生成するためのIP（インストラクションポインタ）と四つのセグメントレジスタ（CS, SS, DS, ES）を含み，これらのものを合成することにより実際のアドレスの生成を行う．

実行ユニット（EU）は，アキュムレータ（Aレジスタ）および，汎用のレジスタ群を含んでおり，前述の命令キューから取り出した命令コードを命令デコーダにセットし，その解読結果に基づき命令の実行動作を行う．レジスタ群は，8080/8085のレジスタとの互換性を考慮するとともに，それをレジスタペアとして16ビットとして扱えるよう構成している．

従来のものに追加されたものとして，Aレジスタの上位8ビット（AH）およびフラグレジスタの上位8ビットのほか，新たに加わった繰返しおよびストリング動作等の命令を可能にするSI，DIおよびBPレジスタがある．

アドレスバス
(20ビット)
データバス
(16ビット)
汎用レジスタ
AH AL
BH BL
CH CL
DH DL
SP
BP
DI
SI
CS
DS
SS
ES
IP
内部コミュニケーションレジスタ
ALUデータバス
(16ビット)
BUSコントロールロジック
8086バス
一時的レジスタ
ALU
EUコントロールシステム
Q BUS
(8ビット)
命令キュー
1 2 3 4 5 6
(8088は4まで)
フラグ
実行ユニット
(EU)
バスインタフェースユニット
(BIU)

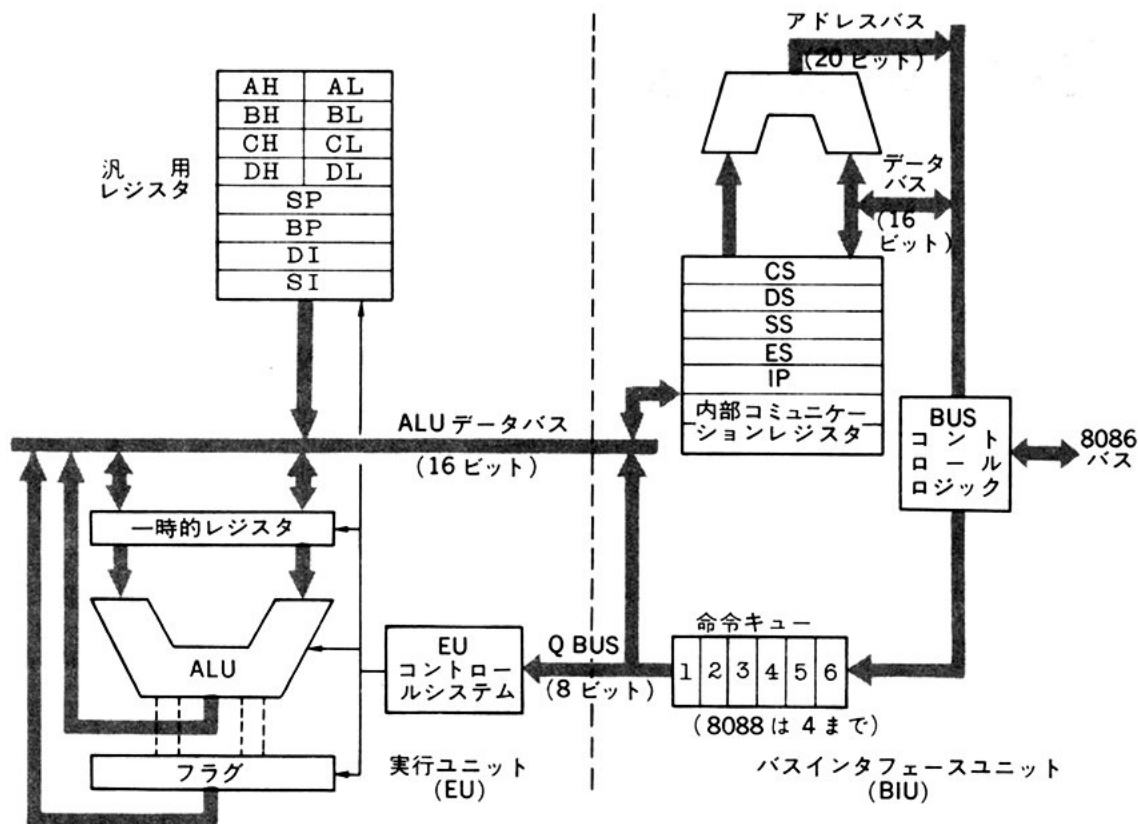



図 2・1　8086の基本的ブロックダイヤグラム

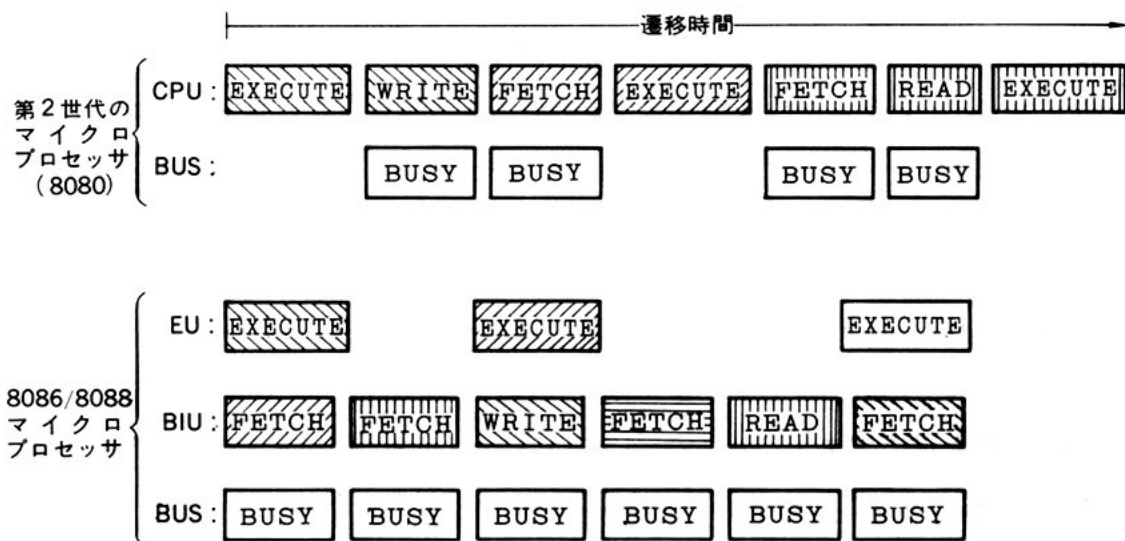



図 2・2　命令のフェッチと実行（バスの使用頻度）

2・2 レジスタの構成

レジスタはCPUが各種命令を実行する際，データや計算の中間結果の一時的な保持，プログラム中でのカウンタとしての使用など，汎用的に使用できるRAMで，バスインタフェースの内部バスに直接つながっているためCPUからのアクセスタイムは外部メモリに比べて短く，指定方法も簡単である．

図2・3は8086/8088レジスタをまとめたもので，斜線の部分は従来の8080/8085レジスタと共通した部分で，その他は新しく追加されたものである．ただ従来の呼び名と多少変わっており，図の左側に示すのが8080系統の呼び名である．

このレジスタは機能的にさらに四種類に分類できる．すなわち，i）汎用レジスタ，ii）セグメントレジスタ，iii）命令ポインタ，iv）フラグレジスタである．

〔1〕 **汎用レジスタ**　汎用レジスタAX，BX，CX，DXはそれぞれ16ビットのレジスタであるが，これを上位8ビットおよび下位8ビットに分けて，8ビットのレジスタとして扱うことも可能で，命令のオペランドの指定により使い分ける．**AX**は**Aレジスタ**または**アキュムレータ**とも呼ばれ，CPUとメモリや外部装置との間のデータのやり取り，各種演算動作等に必ず使用される最も重要なレジスタである．これらのレジスタは，ある種の命令では，暗黙のうちに表2・1に示すような用途に使用される．

B（ベース）**レジスタ**はAレジスタの拡張および補助的に，**Cレジスタ**はカウンタ的用途で，**Dレジスタ**はデータ用に使用するよう統一されている*．SPは**スタックポインタ**で，サブルーチンコール（CALL命令）または割込みの際の返り番地格納用のRAMメモリを指示するポインタ，SIおよびDIは今度8086の命令で強化されたストリング動作の際のソースインデックス（ソースデータの指示）およびディスティネーションインデックス（宛先）である（詳細は6章参照）．

〔2〕 **命令ポインタ**　従来のプログラムカウンタ（PC）に相当するものであるが，8086の場合はこれ単独では命令をフェッチするアドレスにはならず，この後に述べるセグメントレジスタと加算されてアドレスの生成を行う．

---

* 用途については特に限定されていないが，このように統一をとっておくのが望ましい．

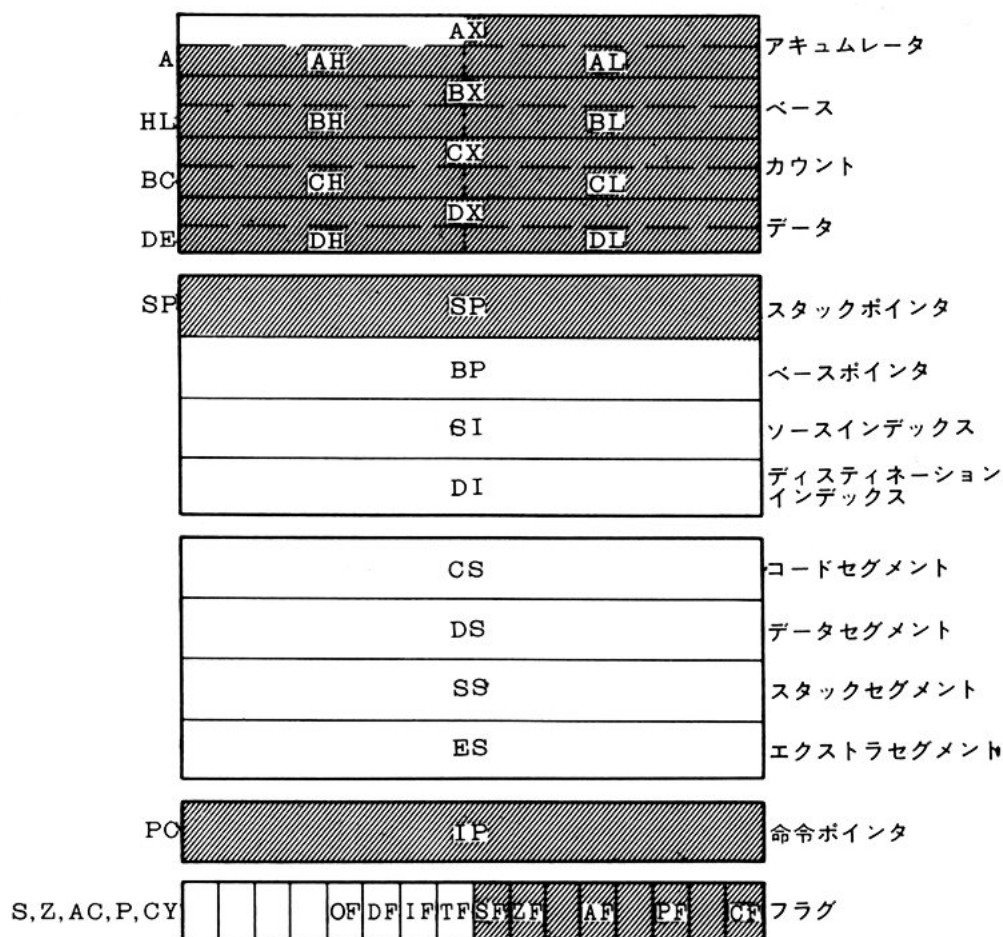



図 2・3　8086/8088 のレジスタ構成

表 2・1　論理アドレスのソース

メモリ参照のタイプ	デフォルトのセグメントベース	代替のセグメントベース	オフセット
命令のフェッチ	CS	なし	IP
スタック動作	SS	なし	SP
変数(下のものを除く)	DS	CS, ES, SS	実効アドレス
ストリングソース	DS	CS, ES, SS	SI
ストリングディスティネーション	ES	なし	DI
ベースレジスタとして使われるBP	SS	CS, DS, ES	実効アドレス

〔3〕 **セグメントレジスタ**　8086の1Mバイトのアドレス空間は，各64Kバイトの論理セグメントに分割されている．CPUは図2·3に示すような4個のセグメントレジスタを持っており，いくつかの命令を使って，それらの操作が可能である．この4個のセグメントの基本的な働きを次に説明する．

（a） **CSレジスタ**　コードセグメントレジスタはプログラムコードを指定するためのレジスタで，インストラクションポインタ（IP）と加算されて（2·3節参照），次にフェッチする命令のアドレスを発生する．CS＝0の場合はIPだけで決定され，従来のPC＝IPと考えられる．

（b） **DSレジスタ**　データセグメントは，プログラム中のデータ部分をアクセスするのに主に使用され，命令中の実効アドレスと加算されてアクセスするデータを指定する．またストリング動作の場合はソースインデックス（SI）と加算されて，そのソースデータを指定する．

（c） **SSレジスタ**　スタックセグメントは，サブルーチンコールや，割込みの際のCS，IPおよび他のレジスタ等を退避するメモリアドレス（RAM）を，SPと加算されて発生する．動作は従来のスタックポインタの場合と同じである．

（d） **ESレジスタ**　エクストラセグメントもデータストレージの指定に使用され，ストリング動作時に，ディスティネーションインデックス（DI）と加算されて，そのディスティネーション（宛先）データの指定を行う．

以上の各セグメントの使用とそのオフセットとの関係を図2·4に示す．

命令により，これらの四つのセグメントレジスタを使ってそれぞれのメモリ空間をアドレス可能で，図2·5に示すように，各セグメントにより指定される物理的メモリロケーションは，オーバラップすることもできる．2·3節で述べるように，セグメントレジスタは，インストラクションポインタIPと加算する際に，4ビット分上位にシフトして行うので，16バイト間隔でそのセグメントの開始アドレスの指定ができる．通常各セグメントレジスタはプログラムの最初で定義され，以後はそれについて何らわずらわされることなくプログラムを書くことができる．CS＝DS＝SS＝ES＝0000Hの場合はアドレッシングはIPおよび命令のオペランドのみで決まり従来の80/85等のプログラムカウンタPCと等価になる．

メモリ　アドレス

物理アドレス

論理アドレス

セグメント
ベース

オフセット
(10H)

110H
10FH
10EH
10DH
10CH
10BH
10AH
109H
108H
107H
106H
105H
104H
103H
102H
101H
100H

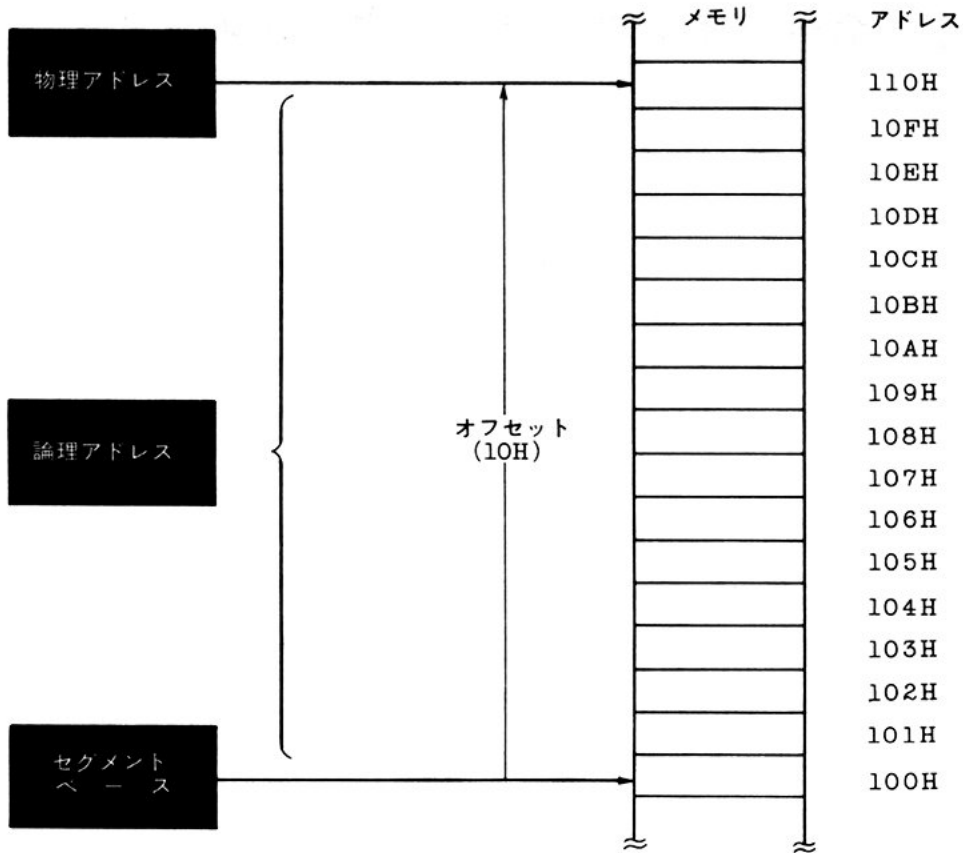



図 2・4　論理アドレスと物理アドレス

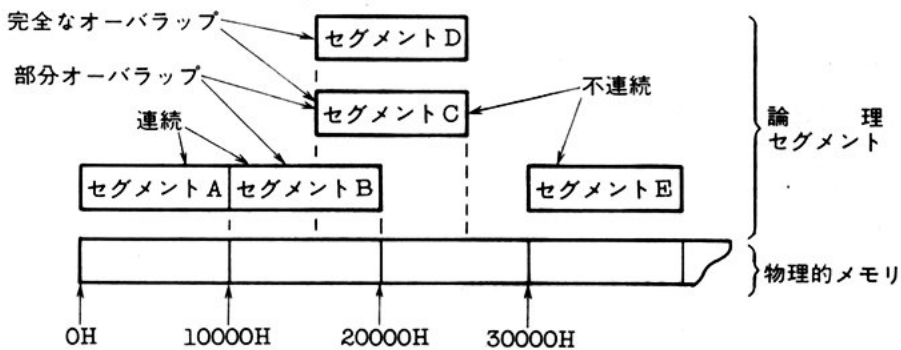



図 2・5　セグメントレジスタの使用

2・3 命令ポインタ(IP)とアドレスの生成

命令ポインタは従来のプログラムカウンタPCに相当し，16ビットで構成されているので0～64Kバイトまで，IPだけで指定できる．これを前記の四つのセグメントレジスタと加算して，1Mバイトまでのメモリ空間の指定を可能にしている．各セグメントレジスタは，プログラムによりそのセグメントのベース(始まり)値が前もって設定されており，コードではIPの値が，データでは命令のオペランドで指定された値が，スタックではSPの値が，それぞれ加算されて実際の物理アドレスとなり，CPUのBIUのアドレスラインから出力される．

図2・6，図2・7に各セグメントレジスタと16ビットのオフセットを加算して実際のアドレスを生成する様子を示す．この方法の特徴は，加算する際にセグメントレジスタを4ビット分左にシフトすることで，セグメントベースは実際のメモリ上では16バイト跳びになる．ただし，これはあくまでもそのセグメントの始まりのことで，最終アドレスはオフセットで調整され，連続的な指定が可能になる．

それぞれの動作によって四つのセグメントレジスタのうちのどれを使用するか，加算するオフセットとして何が使用されるかは，表2・1(前節)を参照．命令のフェッチの場合はCSとIPに，そしてスタック動作はSSとSPに決まっている．変数参照の場合は，通常はセグメントレジスタとしてDSが使用されるが，プログラム指定(ASSUME指令)によりCS, ESおよびSSによる指定も可能である．

ストリング動作の場合のソースデータの指定は**デフォルト***としてはDSであるが，その他にCS, ESおよびSSの使用が可能である．ただし，ディスティネーション指定の場合はESだけである．この場合のオフセットは，ストリングソースはSIが，ストリングディスティネーションにはDIが使用される．

BP(ベースポインタ)が命令中で，ベースポインタとして指定された場合は，デフォルトとして指定されるデータはSS(スタックセグメント)にあるものとして指定される．これも前と同様に，ASSUME指令により指定することにより，CS, DSおよびESの使用も可能である．

* 特に何も指定しない場合に使用されるもの．

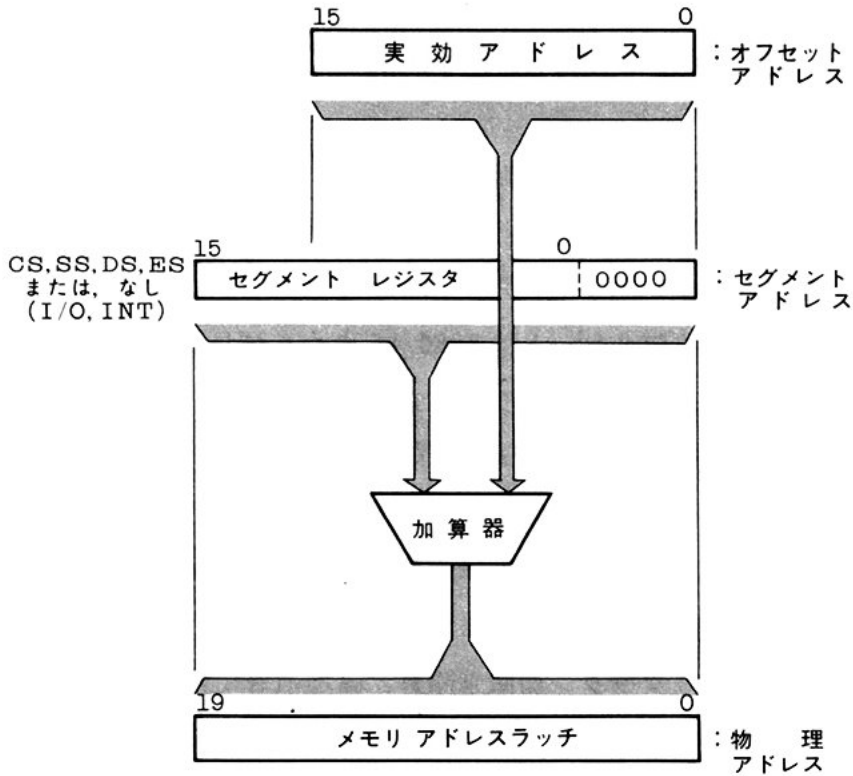



図 2・6 アドレスの生成

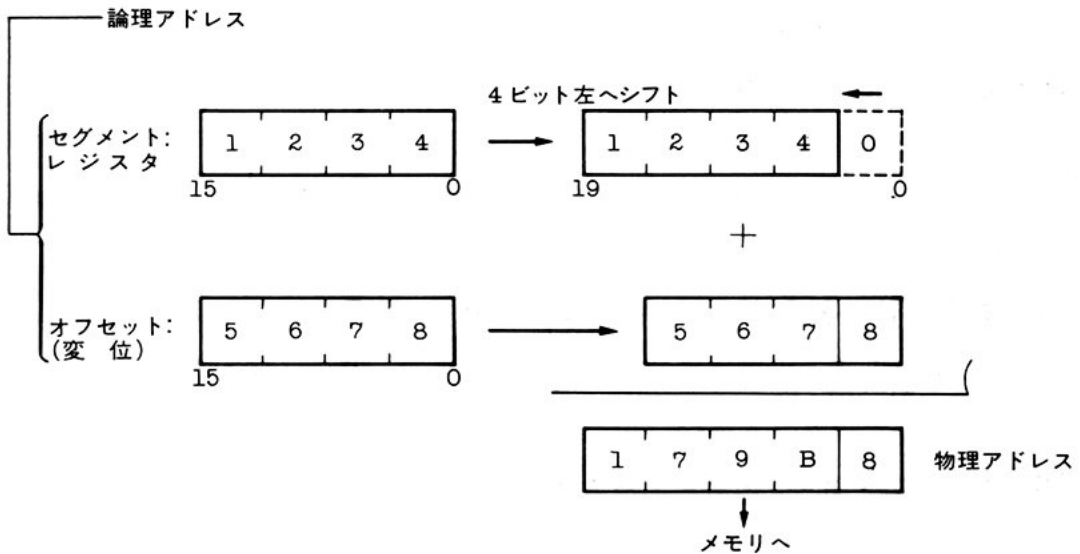



図 2・7 アドレスの生成の例

2・4 フラグの構成と使用

フラグはCPUが各種の演算や動作をした場合の結果としてセットされるもので，後でプログラムによりフラグを調べることによってCPUの動作状態をチェックできる．フラグの構成を図2・8に示す．8086ではフラグレジスタとして1ワード分使われ，そのうち下位8ビットは8080/85と全く同じで，上位に4ビット新しいフラグが追加された．次に各フラグの意味を説明する．

CF（キャリー）フラグ：CPUの加減乗除算，論理演算，ローテート等の動作により，最上位ビットからの桁上げまたは桁下げが生じたことを示す．

PF（パリティ）フラグ：データの伝送等に伴う結果が偶数パリティの場合にセットされる．データ伝送のチェックに使われる．

AF（補助キャリー）フラグ：加減乗除算，論理演算に伴い，8ビット量の**下位ニブル**[*1]から**上位ニブル**への桁上げ，あるいは高位ニブルから下位ニブルへの桁下げが発生した．このフラグは10進演算の場合の補正に使用される．

ZF（ゼロ）フラグ：各演算に伴う結果がゼロの場合にセットされる．

SF（サイン）フラグ：各演算に伴う結果の最上位ビットが1のときセットされる．2進数は2の補数表示で表され，SFは結果の符号(0：正，1：負)．

OF（オーバフロー）フラグ：各演算結果によりオーバフローが発生したことを示す．結果の最上位桁は失われる．

以上のものはCPUの各動作に伴う状態を示すもので，**状態フラグ**と呼ばれる．8086/88では，これにさらに三つの**コントロールフラグ**が追加されている．

IF（割込みイネーブル）フラグ：**外部マスカブル割込み**[*2]を可能にする．このフラグをリセットすると割込み禁止となる．ただし，ノンマスカブル割込み(NMI)，内部割込みには影響を与えない．

DF（ディレクション）フラグ：**ストリング動作**[*3]の場合に，ソースまたはディスティネーションアドレスを自動的に1ずつ増加/減少させる．DFフラグをクリ

---

*1　4ビット単位のことで，8ビットの場合のバイトに対比．

*2　マスクすることが可能な割込みで，INTR端子の割込み．

*3　連続したバイト/ワードを扱う動作．

ヤするとオートインクリメント，すなわちストリングを左から右に処理する．

TF（トラップ）フラグ：プロセッサをシングルステップ状態にする．このモードでCPUは1命令ごとに内部割込みを発生し，実行結果を調べることができる．

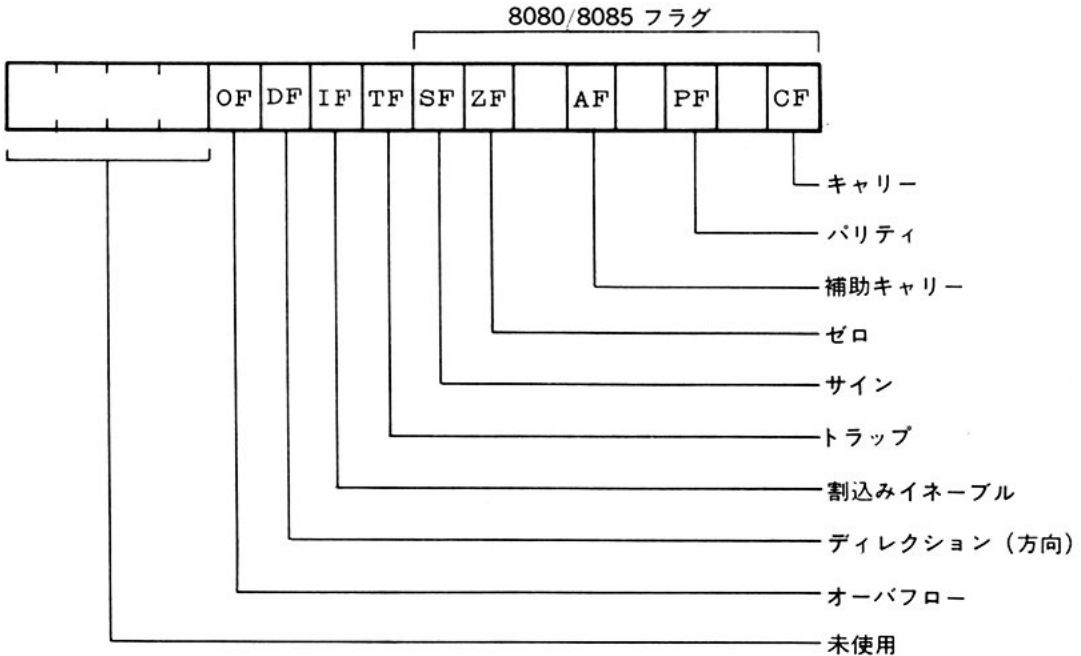



図 2・8 フラグレジスタの構成

2・5　アドレスバスおよびデータバスの構成

8086/8088は図2・9に示すように三つのバス，すなわちアドレスバス，データバスおよびコントロールバスから構成されている．

アドレスバスはA_0～A_{19}の20本で構成され，1Mバイトまでのアドレス空間を持っている．A_0～A_{15}（8088の場合はA_0～A_7）は8085と同様データバスと共用になっており，時間的に切り換えて使用する（図2・10参照）．また上位A_{16}～A_{19}はステイタス情報S_3～S_6と共用になっており，現在データをアクセスするのにどのセグメントレジスタが使用されているかを示している．

8086/8088では33番ピンのMN/$\overline{\text{MX}}$の論理レベルを切り換えると**ミニマムモード**（HIGH）または**マキシマムモード**（LOW）になり，コントロールラインの使用法が異なっている．すなわち，ミニマムモードの場合にはすべてのコントロール信号はCPUから直接出力されるが，マキシマムモードではCPUから出力されるS_0～S_2の信号を8288バスコントローラ（9章参照）が受け取り，それをデコードすることによりそれらの信号を発生する．バスの構成法には基本的には二つの方法がある．一つは図2・9のようにマルチプレクスされたバスをそのまま使用する方法，もう一つは双方向性のバスバッファを入れる場合である（図2・12参照）．最初の方法ではCPUのバスがそのまま外部素子に接続されているので，アドレスをデコードした信号だけでその素子をデータバスに接続するようにコントロールするとデータバスの状態を乱すことになり*，命令フェッチの動作が正常に行われないので注意を要する．チップセレクト端子以外に出力コントロール（OEなど）をもっている素子の場合は，READ信号によりこの端子をコントロールして前記の問題を解決する．この場合のもう一つの注意事項は，8086のドライブ能力の問題で最大許容シンク電流および負荷容量はそれぞれ2mAおよび100pFである．データバスにバッファを挿入する場合，8286/8287双方向性**バスドライバ**があり，このバスの方向の切換えにはCPUからDT/$\overline{\text{R}}$が，そのイネーブルのためにはDENが供給されており，ドライブ能力は32mAおよび300pFに拡張される．

---

* アドレスの出力と同時に選ばれ，命令フェッチと競合するため．

図 2・9 マルチプレクスされたデータバス

T_1 T_2 T_3/T_w T_4
CLK
A_{19}/S_6, A_{16}/S_3 アドレス ステイタス
READY
AD_{15}-AD_0 アドレスA_{15}-A_0 フローティング データ入力 D_{15}-D_0 フローティング
リードサイクル RD DT/R DEN
AD_{15}-AD_0 アドレス データ出力
ライトサイクル WR DEN DT/R

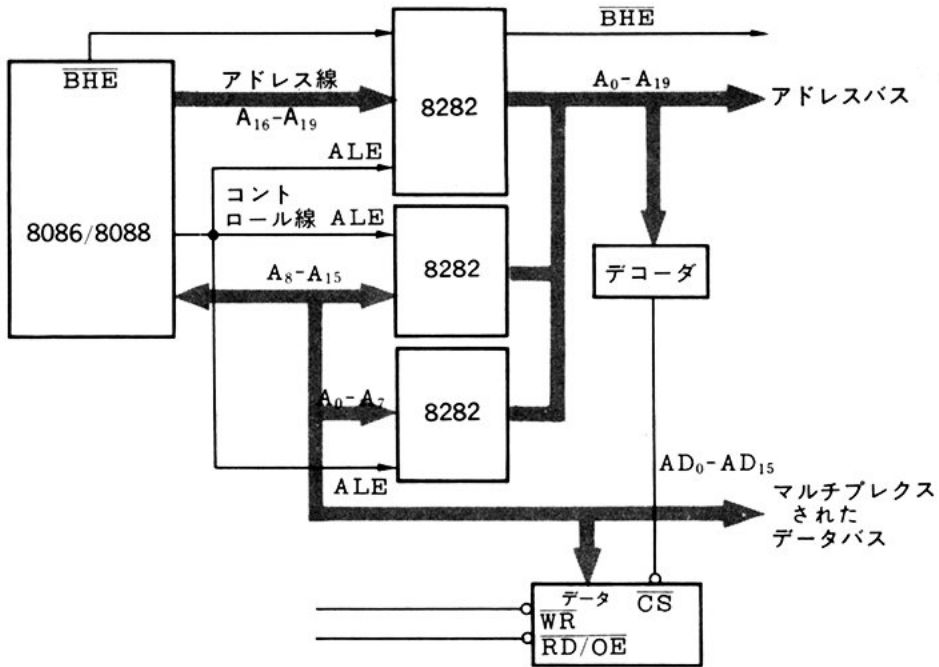
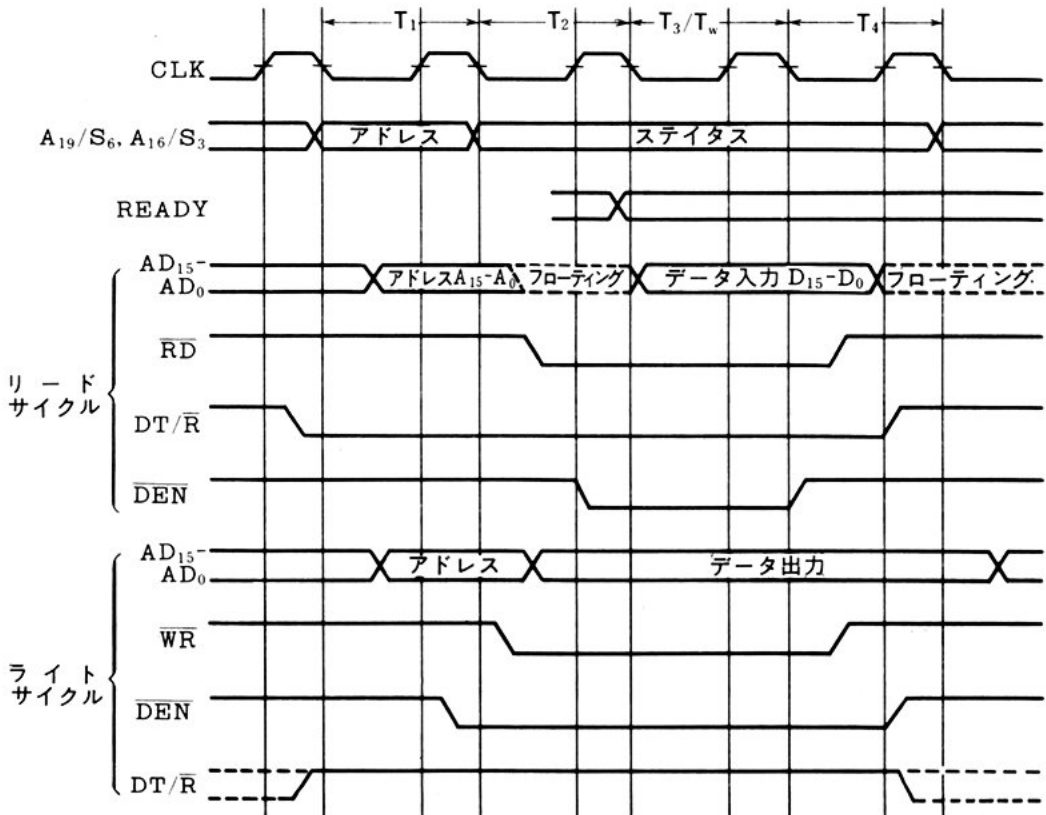



図 2・10 8086 基本バスタイミング

2・6 MAX/MIN モード

8086/8088は小規模システムから，マルチプロセッサによる大がかりなシステムまでをサポートできるように，33番ピンを+5V（ミニマムモード）にするか，0V（マキシマムモード）にするかで，二つの動作モードを選べるようになっている．

ミニマムモードの場合，CPUからすべてのバスコントロール信号が供給され，少ないチップ構成でシステムの構成が可能である．DMA動作のためのHOLD/HOLDA信号も直接CPUから供給され，信号のタイミングも従来の80/85の場合と同じで，8237/8257等のDMAコントローラも使用可能である．

マキシマムモードでは8288バスコントローラと組み合わせて使用する．状態情報としてCPUから$\overline{S_0}$～$\overline{S_2}$が供給され，それを8288がデコードしてメモリおよびI/Oのリード/ライト等のコントロール信号を発生する．また8289バスアービタと8288の組合せでマルチバス適合のシステムバスも構成可能である．

ミニマムモードのHOLD/HOLDAは，マキシマムモードでは2本のリクエスト/グラント（$\overline{RQ}/\overline{GT_0}$および$\overline{RQ}/\overline{GT_1}$）に置き換えられる．このリクエスト/グラントシーケンスは，要求，認可，および解放の三つのフェーズから構成されている（詳細は4・2節参照）．まず，バスの使用を要求するプロセッサが$\overline{RQ}/\overline{GT}$線にパルスを送ることで始動され，次にCPU側から同じ信号線を使って，システムバスがフローティングになったこと，および次のグラントフェーズでバスコントローラから論理的に切り離されることを，要求中のプロセッサに知らせるため，パルスを出力し，ホールド状態に入る．最後に，その動作の終了として，要求中のプロセッサが$\overline{RQ}/\overline{GT}$線にパルスを出力し，バスを解放する用意があることをCPUに知らせ，次のクロックサイクルでCPUが再びバスのアクセス権を取り戻す．また8289バスアービタと結合して，共有システムバスをある命令の間独占的に使用することを保証する$\overline{LOCK}$信号を，命令の前に付加する**プリフィックス**という1バイトの命令により，ソフトウェアコントロール可能である．またキュー状態情報QS_0およびQS_1はICE-86エミュレータや8087コ・プロセッサがCPUの命令実行過程を追跡できるよう出力されている（10，12章参照）．

共通信号		
名称	機能	形式
AD_{15}-AD_0	アドレス/データバス	双方向3ステート
A_{19}/S_6-A_{16}/S_3	アドレス/ステイタス	出力3ステート
$\overline{BHE}/S_7$	バスハイイネーブル/ステイタス	出力3ステート
$MN/\overline{MX}$	マキシマム/ミニマムモード	入力
$\overline{RD}$	リードコントロール	出力3ステート
$\overline{TEST}$	テスト/ウェイト	入力
READY	レディコントロール	入力
RESET	システムリセット	入力
NMI	ノンマスカブル割込み	入力
INTR	割込み要求	入力
CLK	システムクロック	入力
V_{CC}	+5V	入力
GND	グランド	

ミニマムモード信号 ($MN/\overline{MX}=V_{CC}$)		
名称	機能	形式
HOLD	ホールド リクエスト	入力
HOLDA	ホールド アクノレージ	出力
$\overline{WR}$	ライト コントロール	出力3ステート
$M/\overline{IO}$	メモリ/IOコントロール	出力3ステート
$DT/\overline{R}$	データ伝送/受信	出力3ステート
$\overline{DEN}$	データイネーブル	出力3ステート
ALE	アドレスラッチイネーブル	出力
$\overline{INTA}$	割込みアクノレージ	出力

マキシマムモード信号 ($MN/\overline{MX}$=GND)		
名称	機能	形式
$\overline{RQ}/\overline{GT}_{1,0}$	リクエスト/グラントバスアクセスコントロール	双方向
$\overline{LOCK}$	バス優先ロックコントロール	出力3ステート
$\overline{S_2}$-$\overline{S_0}$	バスサイクルステイタス	出力3ステート
QS_1, QS_0	命令キューステイタス	出力

8086 CPU

信号	ピン	ピン	信号	(マキシマムモード)
GND	1	40	V_{CC}	
AD_{14}	2	39	AD_{15}	
AD_{13}	3	38	A_{16}/S_3	
AD_{12}	4	37	A_{17}/S_4	
AD_{11}	5	36	A_{18}/S_5	
AD_{10}	6	35	A_{19}/S_6	
AD_9	7	34	$\overline{BHE}/S_7$	
AD_8	8	33	$MN/\overline{MX}$	
AD_7	9	32	$\overline{RD}$	
AD_6	10	31	HOLD	($\overline{RQ}/\overline{GT_0}$)
AD_5	11	30	HOLDA	($\overline{RQ}/\overline{GT_1}$)
AD_4	12	29	$\overline{WR}$	($\overline{LOCK}$)
AD_3	13	28	$M/\overline{IO}$	($\overline{S_2}$)
AD_2	14	27	$DT/\overline{R}$	($\overline{S_1}$)
AD_1	15	26	$\overline{DEN}$	($\overline{S_0}$)
AD_0	16	25	ALE	(QS_0)
NMI	17	24	$\overline{INTA}$	(QS_1)
INTR	18	23	$\overline{TEST}$	
CLK	19	22	READY	
GND	20	21	RESET	

〔注〕 カッコ内はマキシマムモード

図 2・11 ミニマムモード/マキシマムモードと各信号

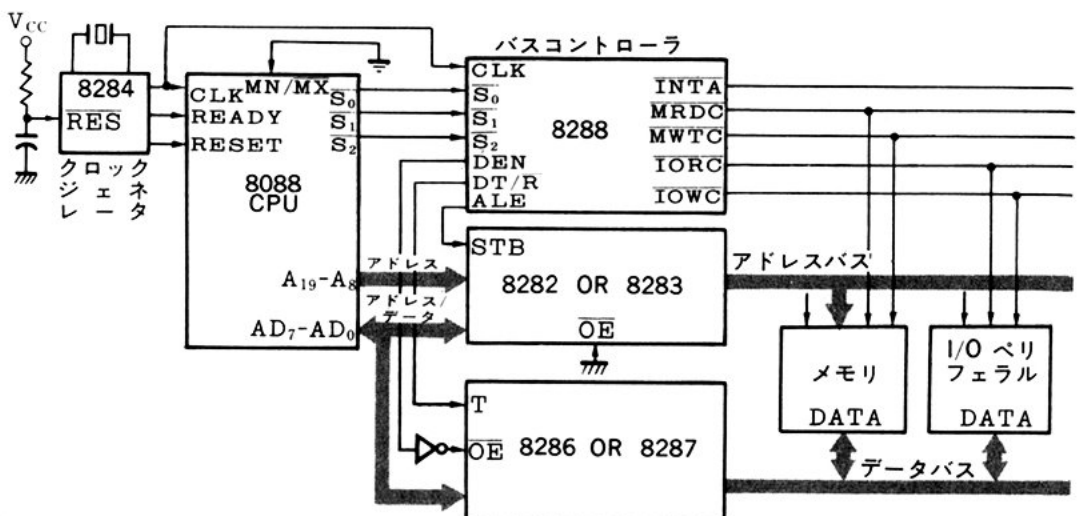



図 2・12 マキシマムモードのシステム例

ミニマム/マキシマムモードの信号比較一覧

8086		ピン番号	8088	
動作モード			動作モード	
ミニマム	マキシマム		ミニマム	マキシマム
HOLD	$\overline{RQ}/\overline{GT0}$	31	HOLD	$\overline{RQ}/\overline{GT0}$
HOLDA	$\overline{RQ}/\overline{GT1}$	30	HOLDA	$\overline{RQ}/\overline{GT1}$
$\overline{WR}$	$\overline{LOCK}$	29	$\overline{WR}$	$\overline{LOCK}$
$M/\overline{IO}$	$\overline{S2}$	28	$IO/\overline{M}$	$\overline{S2}$
$DT/\overline{R}$	$\overline{S1}$	27	$DT/\overline{R}$	$\overline{S1}$
$\overline{DEN}$	$\overline{S0}$	26	$\overline{DEN}$	$\overline{S0}$
ALE	QS0	25	ALE	QS0
$\overline{INTA}$	QS1	24	$\overline{INTA}$	QS1
		34	SS0	High State

3. メモリの構成

20本のアドレス線により1Mバイトまで拡張されたメモリ構成について述べ，メモリとのインタフェースについて考察する．また，サブルーチンコールおよび割込みの場合のスタックレジスタの使用と動作についても記述している．

3・1 メモリの構成と使用

8086 システムのメモリ構成は，物理的配置としては 16 ビットを 1 ワード単位とした 0～512K ワードになっているが，CPU から指定される論理アドレスは 0～1M バイトの連続したメモリとして扱われ，ワードデータは連続したバイトで構成される．そのメモリ構成図を図 3・1 に示す．

このように構成されたメモリをアクセスするためには，CPU は A_{19}～A_1 により各ワードの選択をし，A_0 と $\overline{BHE}$（バス ハイイネーブル）の組合せにより，それぞれその下位バイトまたは上位バイト，およびその両方の選択を行う．図 3・2～図3・5 にその選択の模様を示す．A_0, $\overline{BHE}$ ともにアクティブ LOW の信号で，各バイトまたはワードの指定は次のようにして行う．ここでは A_{19}～A_1 により X+1, X が指示されているものとする．

(1) 偶数アドレスのバイト転送の場合：A_0＝LOW, $\overline{BHE}$＝HIGH で，D_7～D_0 に (X)* が現れる．

(2) 奇数アドレスのバイト転送の場合：A_0＝HIGH, $\overline{BHE}$＝LOW で，D_{15}～D_8 に (X+1) が現れる．

(3) 偶数アドレスのワード転送の場合：A_0＝LOW, $\overline{BHE}$＝LOW になり，D_{15}～D_0 に偶数アドレスのワード値 (X+1), (X) が現れる．

(4) 奇数アドレスのワード転送の場合：最初のバスサイクルで A_0＝ HIGH, $\overline{BHE}$＝LOW になり，(X+1) が D_{15}～D_8 に現れ，次のバスサイクルで，A_0＝LOW, $\overline{BHE}$＝HIGH になり (Y) が D_7～D_0 に現れる．たとえば奇数アドレスのメモリロケーションから C_L レジスタにバイトデータをロードする場合，そのデータは データバスの上位 8 ビットを 通して 8086 中に 転送され，自動的に 8086 の内部 16 ビットデータバスの 下位 8 ビットに 指し向けられ，それから C_L レジスタ中に格納される．

* ( ) はそのメモリの内容を表す．

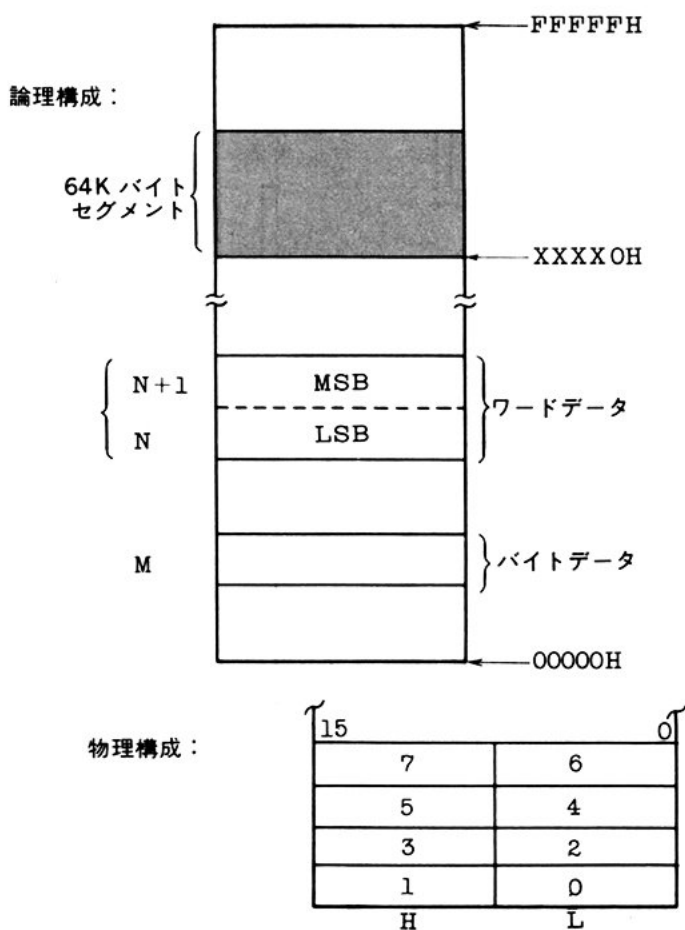



図 3・1 メモリの構成

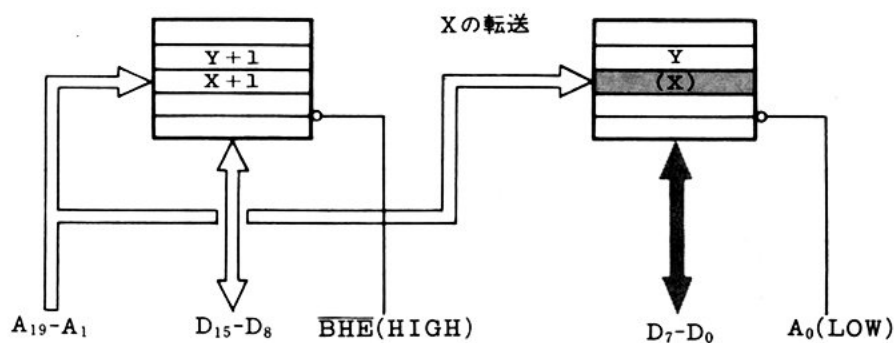



図 3・2 偶数アドレスバイトの転送

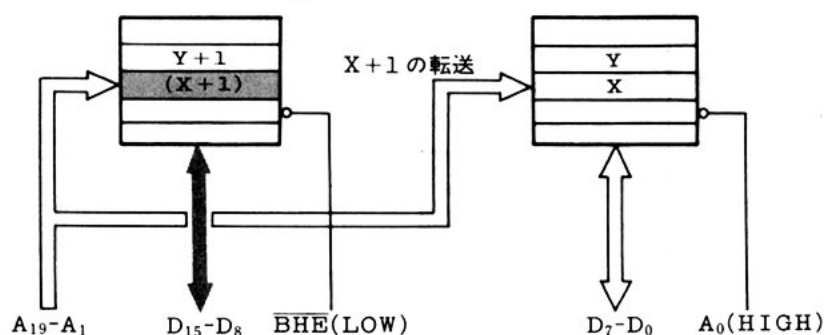



図 3・3 奇数アドレスバイトの転送

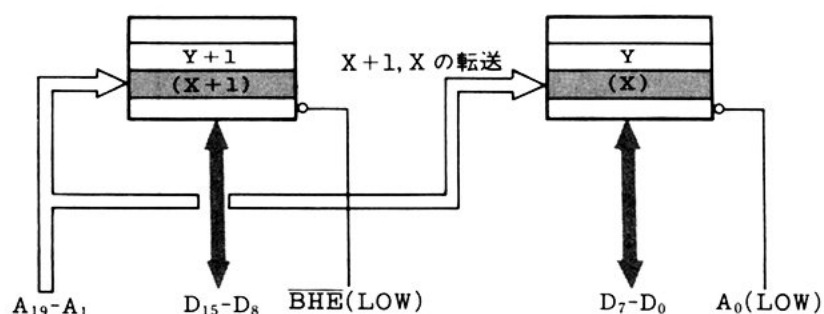



図 3・4 偶数アドレスのワード転送

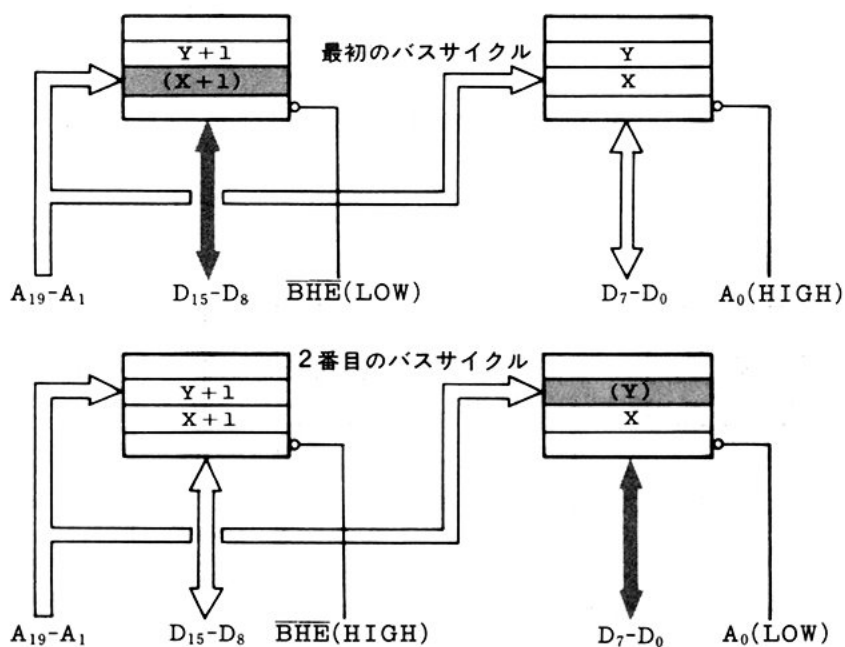



図 3・5 奇数アドレスワード転送

3・2 メモリのセグメンテーション

8086/8088の1Mバイトのメモリ空間は四つのセグメントレジスタCS, DS, SS, ESにより分割され，オフセットのIP(命令コードの場合)または命令のオペランドとの組合せで64Kバイトのモジュールとしてのアドレッシングが可能である．各セグメントレジスタの値はそのセグメントの始まりを表し，それにオフセットを加算した値が実際のアドレスとなる．

セグメントレジスタはオフセットと加算する場合4ビット上位にシフトして行うので，このセグメントベースは16バイト跳びで0～FFFFFHのどこにでも持っていける(2・3節)．これらのセグメントは図3・6のように，連続，部分的重なり，全体的重なり，不連続と，任意の組合せで使用できる．それゆえ一つの物理的メモリロケーションの複数の論理セグメントへの割当ても可能である．

これらのセグメントレジスタは通常，プログラムの最初の部分で必要な値に初期化され，その後はそれについて考えなくてもよい．図3・7にアセンブラでのセグメントのセットアップのプログラム例を示す．この中で，SEGMENT指令がセグメントの始まりとなりEND指令で終わる．この間に書かれた命令およびデータが，その名前のセグメントに属するものになる．完全な一つのプログラムを一つのセグメント中で書くことが可能であるが，この場合はすべてのセグメントレジスタの値は同じベースアドレスを持ち，このメモリセグメントは完全にオーバラップする．

一方，非常に大きなプログラムの場合は，一つのプログラムが多数のセグメントに分割可能で，通常プログラム中の最初の命令で，セグメント名とセグメントレジスタの対応を設定し，それからその対応しているセグメントのベースアドレスを各セグメントレジスタにロードする．アセンブラのASSUME指令が実行時に，どのアドレスがそのセグメントレジスタ中に入るかを伝える．

その仮定されたレジスタが，その命令のタイプに対してハードウェアが予想しているレジスタ（表2・1参照）である場合は，定められたとおりの機械語命令をアセンブラは発生する．それに対し，ハードウェアがあるレジスタが使用される

ものと予想しているときに，そのオペランドがそのレジスタにより指定されるレジスタ内にない場合は，アセンブラは自動的にセグメントオーバライドプリフィックスバイトをその機械語コードの前に付加する.

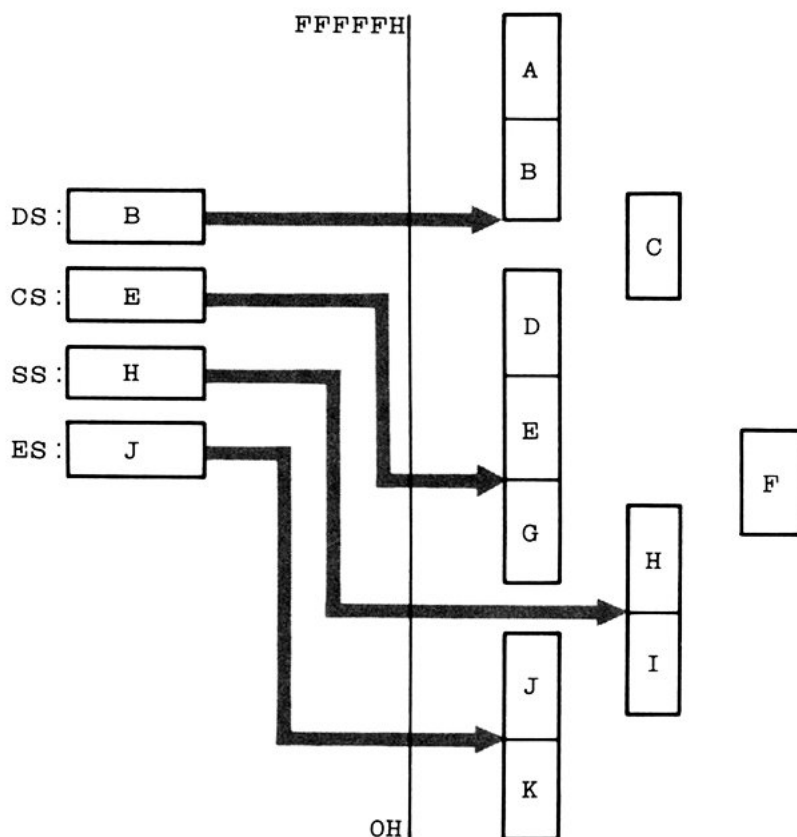



図 3・6 セグメントレジスタの使用

```
DATA_SEG   SEGMENT
   ;DATA DEFINITIONS GO HERE
DATA_SEG   ENDS

STACK_SEG SEGMENT
   ;ALLOCATE 100 WORDS FOR A STACK AND
   ;   LABEL THE INITIAL TOS FOR LOADING SP.
         DW 100 DUP(?)
STACK TOP LABEL WORD
STACK_SEG ENDS

CODE_SEG   SEGMENT
   ;GIVE ASSEMBLER INITIAL REGISTER-TO-SEGMENT
   ;   CORRESPONDENCE. NOTE THAT IN THIS
   ;   PROGRAM THE EXTRA SEGMENT INITIALLY
   ;   OVERLAPS THE DATA SEGMENT ENTIRELY.
ASSUME CS:CODE_SEG,
&      DS:DATA_SEG,
&      ES:DATA_SEG,
&      SS:STACK_SEG

START:;THIS IS THE BEGINNING OF THE PROGRAM.
      ;LOC-86 WILL PLACE A JMP TO THIS
      ;LOCATION AT ADDRESS FFFF0H.

   ;LOAD THE SEGMENT REGISTERS. CS DOES NOT
   ;   HAVE TO BE LOADED BECAUSE SYSTEM
   ;   RESET SETS IT TO FFFFH, AND THE
   ;   LONG JMP INSTRUCTION AT THAT ADDRESS
   ;   UPDATES IT TO THE ADDRESS OF CODE_SEG.
   ;   SEGMENT REGISTERS ARE LOADED FROM AX
   ;   BECAUSE THERE IS NO IMMEDIATE-TO-
   ;   SEGMENT_REGISTER FORM OF THE MOV
   ;   INSTRUCTION.

               MOV AX, DATA_SEG
               MOV DS, AX
               MOV ES, AX
               MOV AX, STACK_SEG
               MOV SS, AX
;SET STACK POINTER TO INITIAL TOS.
                   MOV SP, OFFSET STACK_TOP

;SEGMENTS ARE NOW ADDRESSABLE.
;MAIN PROGRAM CODE GOES HERE.
CODE_SEG                 ENDS

;NEXT STATEMENT ENDS ASSEMBLY AND TELLS
;   LOC-86 THE PROGRAMS STARTING ADDRESS.

                  END START
```

図 3・7 セグメントレジスタのセットアップ例

3・3 スタックの構成と使用

8086/8088のスタックもメモリの場合と同様に，スタックセグメントレジスタ(SS)とオフセットとしてのスタックポインタ(SP)により算出され指定される．そのようにしてRAMにより構成される全メモリ範囲，64Kバイトまで使用可能で，実用上ほぼ無制限といえる．SSは現在使用中のスタックのベースアドレスを保持し，SPはそのスタックの先頭(TOS)アドレスを指示する．

スタックに関する命令は，1度に1ワードずつ，スタック項目に加えたり，取り除いたりする．すなわち図3・8に示すように，SPの値を2減じて，そのセーブする項目をスタック上にプッシュし，新しいTOSのところに書き込まれる．

次にサブルーチンや割込み処理ルーチンからのリターンに際しポップする場合，TOSが指示するメモリ内容をポップしてスタックからその内容をコピーし，SPを2だけ増加させる．この場合のスタックの減少とは，ベースアドレス(SSの値)の方向に近づくということである．ただしスタック動作はけっしてスタック上でその項目を移動させたり消去したりするのではなく，そのスタックの先頭(TOS)がスタックポインタを更新する結果として，変更されるだけである．

具体的には，図3・8において，最初SS＝0105H, SP＝0008Hとすると，スタックの先頭アドレス(TOS)はこの二つを加え合わせた値，すなわち1058Hとなる．次にAX(内容は1234H)をPUSHすると，AXの内容がTOSの示しているアドレスの次(スタック動作では，PUSHはアドレスの高位から低位へ移動することに注意)から下位バイト，上位バイトの順にスタックに退避される．この場合SSの値は不変で，SPの値だけが2バイト分減ぜられて0006Hとなる．

その状態を元に復帰するには(c)のようにPOP AXを実行すると，TOSが指示しているアドレスの内容がAXレジスタに復帰されて，SPの値は0008になる．

次にPOP BXを行うとその次のスタックの内容がBXレジスタに入り，TOSの値は2だけ増加され000AHになる．

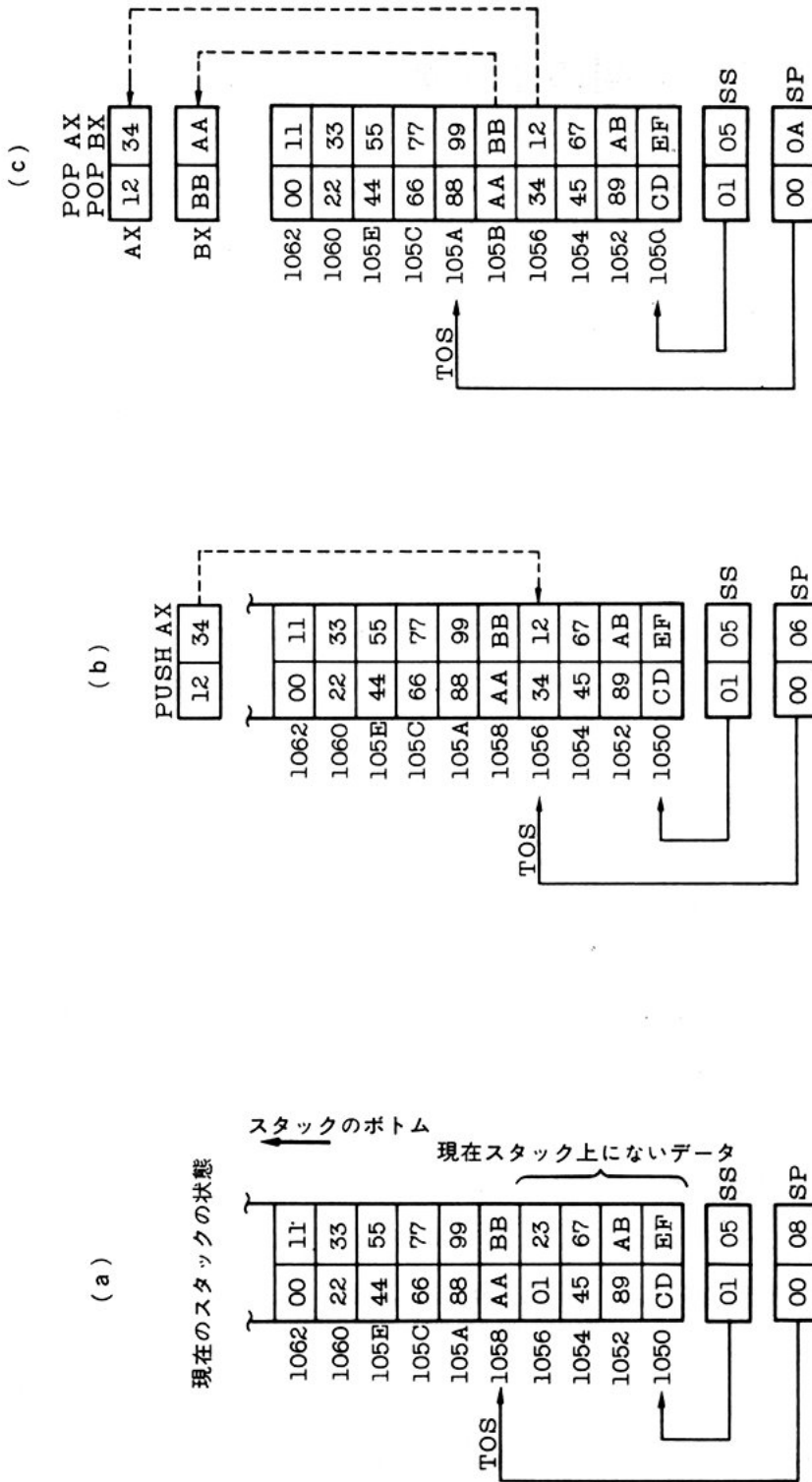
通常の使用は，サブルーチンコール，割込み処理ルーチンの入口でレジスタの退避にPUSHし，次に処理ルーチンの終りでそれらを復帰するのにPOPを行う．

次のコードシーケンスのためのスタック動作
PUSH AX
POP AX
POP BX

図 3・8 スタック動作

3・4 メモリとの間のインタフェース

［1］ **ROM/EPROM の接続**　8086/8088 と ROM/EPROM とのインタフェースの様子を図 3・9 に示す．この場合はリードオンリーメモリであることから，書き換えられる心配がないので，A_0 および $\overline{BHE}$ は使用する必要はなく，バス上のフルワードのうちの 8086 自身が必要とするものだけを読み取る．次に，ミニマム構成の場合のマルチプレクスされたバスに接続されている ROM/EPROMを例にとり，その AC 特性を考察する．

$$TACC = 3TCLCL - TCLAV_{max} - TDVCL_{min} -(\text{アドレスバッファの遅れ})$$

$$= 3\times200\,ns - 110\,ns - 30\,ns - 30\,ns = 430\,ns$$

$$TCE = TACC -(\text{デコーダの遅れ}) = 430\,ns - 18\,ns = 412\,ns$$

$$TOE = 2TCLCL - 195\,ns = 205\,ns$$

$$TDF = 155\,ns$$

ここに，TACC：アドレス出力からデータが有効になるまでの時間(TAVDV)

TCE：チップイネーブルからデータが有効になるまでの時間(TSLDV)

TOE：出力イネーブルからデータが有効になるまでの時間(TRLDV)

TDF：出力イネーブルの HIGH から出力フロートになる時間(TRHDZ)

以上の考察から，使用される EPROM に必要な WAIT ステートの一覧表を表 3・1 に示す．上記の AC パラメータおよびタイムチャートの詳細は付録参照．

［2］ **スタティック RAM**　スタティック RAMのインタフェースには，そのチップセレクト/チップイネーブルのコントロール信号の作成に A_0 および $\overline{BHE}$を含めてデコードしなければならない．図 3・10 には 2114，2141 および 2147 のようにチップイネーブル信号だけをもち，出力イネーブル信号を持たないメモリ素子のためのチップセレクト信号発生の模様を示す．$\overline{BHE}$ で上位バイトを，そして A_0で下位バイトの選択をし，$\overline{RD}$および$\overline{WR}$ の ORで，両チップをイネーブルにする．

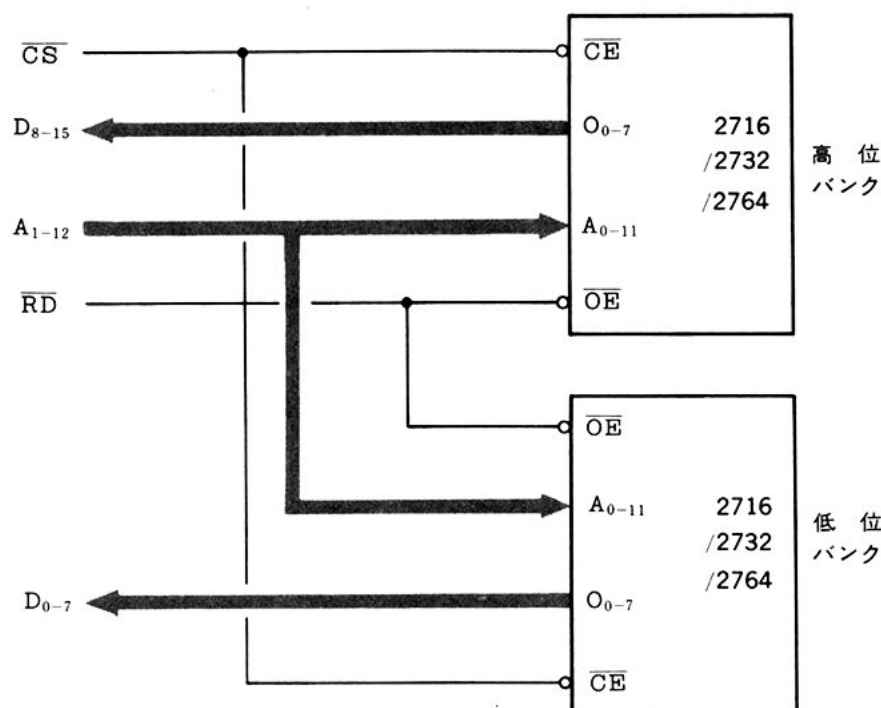



図 3・9 ROM/EPROM のインタフェース

表 3・1 互換EPROMと必要なWAIT数

EPROM 名		ミニマムモード		マキシマムモード	
		バッファなし	バッファあり	バッファあり	完全バッファ付き
2716	450ns	1W	1W	1W	1W
2716-1	390ns	レ*	1W	1W	1W
2716-2	350ns	レ	レ	レ	レ
2732	450ns	1W	1W	1W	1W
2732A	250ns	レ	レ	レ	レ
2732A-2	200ns	レ	レ	レ	レ
2732A-3	300ns	レ	レ	レ	レ
2764	250ns	レ	レ	レ	レ
2764-2	200ns	レ	レ	レ	レ
2764-3	300ns	レ	レ	レ	レ

* WAITなし

次に出力イネーブル端子のある2142のような場合は，図3・11のように$\overline{RD}$でその出力バッファのコントロールができるので，$\overline{WR}$信号を$\overline{BHE}$またはA₀でゲートして，その上位または下位バイトのライトイネーブル信号としている．

図3・12に示す8086マキシマムモードにおける2142のライトタイミングについて考察すると

$$\begin{aligned}
TWA &= 2TCLCL - TCLML_{max} + TCLMH_{min} \\
&= 375\,ns \\
TWR &= 2TCLCL - TCLMH_{max} + TCLLH_{min} + TSHOV_{min} \\
&= 170\,ns \\
TDWA &= 2TCLCL - TCLDV_{max} + TCLMH_{min} - TIVOV_{max} \\
&= 265\,ns \\
TDH &= TCLCH - TCLMH_{max} + TCHDX_{min} + TIVOV_{min} \\
&= 95\,ns
\end{aligned}$$

となり，標準2142は8086システムに完全互換性があり，WAITは必要ない．

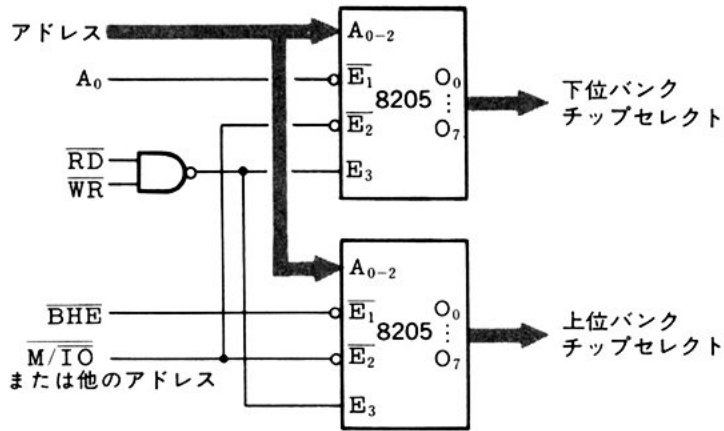



図 3・10　出力イネーブル端子のないメモリ素子のためのデコード

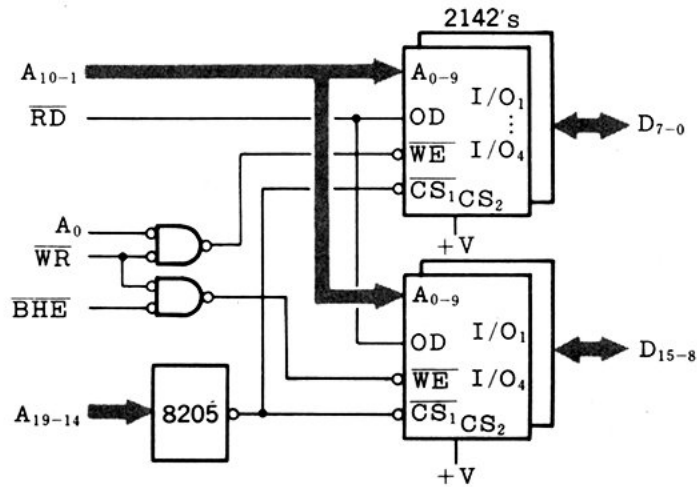



図 3・11　出力イネーブル端子をもつメモリ素子のためのデコード

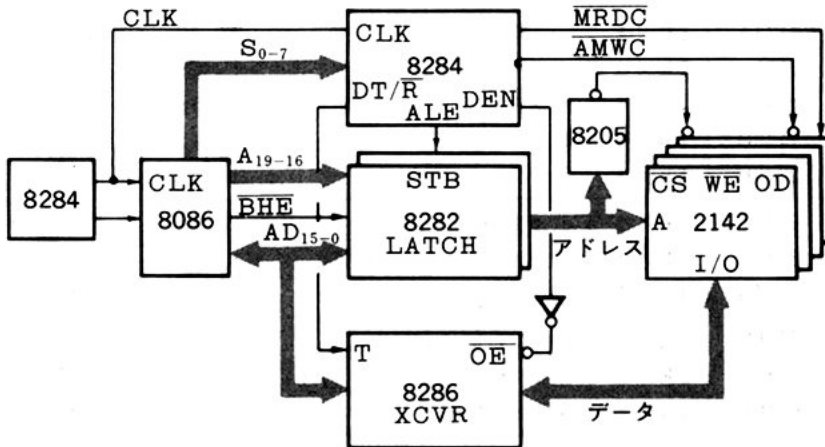



図 3・12　RAM の AC 特性算出の回路例

8086/8088 システム互換のスタティックRAM

製品名	構成	特徴	ピン数	アクセスタイムの範囲〔ns〕
2115A	1K×1	高速-オープンコレクタ	16	45～70
2125A	1K×1	高速-3ステート	16	45～70
2115H	1K×1	高速-オープンコレクタ	16	25～35
2125H	1K×1	高速-3ステート	16	20～35
2114A	1K×4	低消費電力	18	120～250
2142	1K×4	出力イネーブル端子付	20	200～450
2148	1K×4	高速-パワーダウン動作	18	70～85
2148H	1K×4	高速-パワーダウン動作	18	45～55
2149H	1K×4	高速-高速 $\overline{CS}$ 端子	18	45～55
2141	4K×1	低消費電力-パワーダウン動作	18	120～150
2147	4K×1	高速-パワーダウン動作	18	70～85
2147H	4K×1	高速-パワーダウン動作	18	35～55
5516*	2K×8	CMOS-低消費電力	24	250
5517*	2K×8	CMOS-低消費電力，高速 $\overline{CS}$	24	250
6116*	2K×8	NMOS-低消費電力	24	120～200

* 2716EPROMとピンコンパチブル

4. 入力/出力の構成

8086では従来の0～255までのI/Oポートの他に，DXレジスタを使用した間接指定により，0～65Kまでの任意のポートの指定ができる．また，通常のI/O動作のほか，メモリマップとI/OおよびDMA転送についても記述している．

4・1 入/出力動作

8086/8088 のI/O アドレッシングの方法には大別して，I/Oマップト I/Oとメモリマップト I/O の二つがあり，それぞれ異なったアドレス空間を持っている.

〔1〕 **I/Oマップト I/O**　これはメモリアドレス空間とは独立した0〜FFFFH のアドレス範囲を持ち，入出力命令 IN/OUT が使用される．そのうち直接ポート番号をバイト値として指定する場合は0〜255ポートまでの指定が可能である.

```
IN AX,0FFH;INPUT FROM PORT NO=0FFH
OUT 1FH,AX;OUTPUT TO PORT NO=1FH
```

もう一つの方法としては，DX レジスタを使用し，間接アドレッシングとして I/O ポートの指定を行うもので，この場合のI/O指定はDXが16ビットであることから，0〜65535 までの指定が可能になる．命令の実行に先立ち，DX にその I/Oアドレスをセットしておく.

```
IN AX,DX
OUT DX,AX
```

I/O ポートとの間の転送はワード/バイトのいずれも可能で，データバスの上位8ビット/下位8ビットのいずれにも接続でき，バイトデータの場合は通常アキュムレータの下位8ビット（AL）との間で行われる（図4・2参照).

〔2〕 **メモリマップト I/O**　メモリマップト I/O は，I/O 装置をメモリと同等とみなしてアドレスする方法で，その装置がメモリと全く同じ応答をする限り，CPUとしてはこれらを区別する必要はない．このように構成することにより，メモリ空間に配置されているI/Oをメモリ参照命令を使ってアクセスすることができる．たとえば，MOV 命令を使って8086のレジスタとI/Oポート間でデータの転送を行ったり，AND, OR および TEST 命令等を使ってI/O 装置中のレジスタのビット操作を行うことも可能である．しかし，このようにしてI/O装置に割り当てられたメモリ空間は，その分だけメモリ空間を減らすことになるので注意を要する．またメモリ参照命令は IN/OUT のような入出力命令に比較して，実行時間が多少余分にかかるという欠点もある.

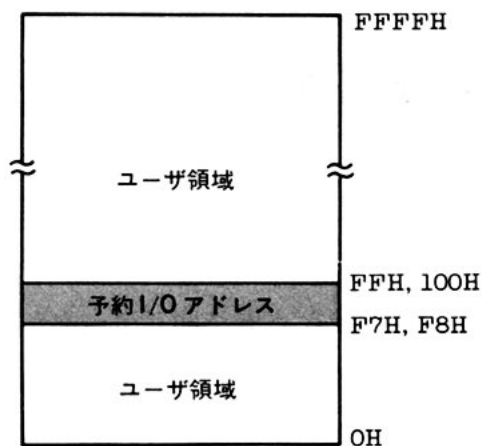



図 4・1 I/Oアドレス空間とその予約されているデバイスアドレス

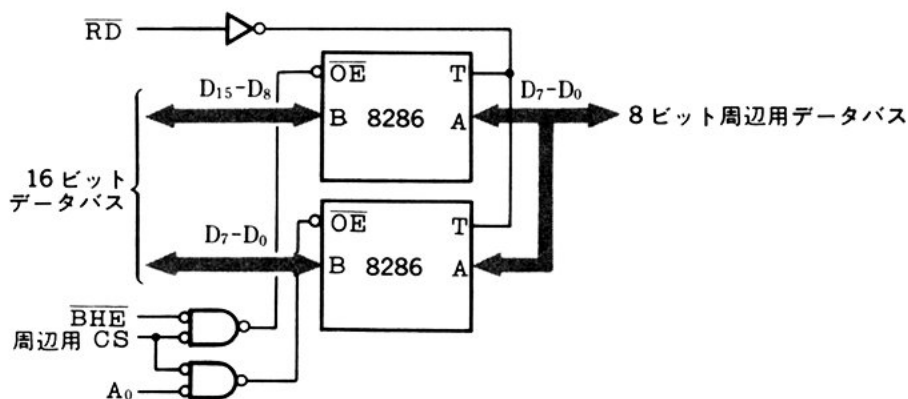



図 4・2 16ビットバスの8ビットバスへの変換

表 4・1 組合せ使用可能な周辺チップ

	構成			
	ミニマムモード		マキシマムモード	
	バッファなし	バッファ付き	バッファ付き	完全バッファ付き
8251A	レ	1W	レ	レ
8253-5	レ	1W	レ	レ
8255A-5	レ	1W	レ	レ
8257-5	レ	1W	レ	レ
8259A	レ	レ	レ	レ
8271	レ	1W	レ	レ
8273	レ	1W	レ	レ
8275	レ	1W	レ	レ
8279-5	レ	1W	レ	レ
8041A	レ	1W	レ	レ
8741A	レ	1W	レ	レ
8291	レ	レ	レ	レ

〔注〕 1. Wは必要なWAITの数.
2. レはWAIT不要を示す.
3. マキシマムモードはバスコントローラ(8288)使用.

4·2 DMA 転 送

8086/8088をミニマムモードにすると8080/8085と同様HOLD·HOLDA信号は直接CPUから提供され，8257，8237等のDMAコントローラを使える.

DMAコントローラは，I/O装置とメモリ間でデータを直接転送するためHOLD要求をCPUに出してバスの使用を要求する.CPUはそれを受け取ると現在実行中のバスサイクルを完了した後，アクノレージ信号HOLDA信号を出して，DMAコントローラのバス使用を許可する.このHOLD信号が取り除かれるまでCPUはバスをフローティングの状態とし，バス使用権を放棄する.

次にマキシマムモードの構成の場合は，2組の$\overline{RQ}/\overline{GT}$信号がHOLD/HOLDAの代わりに使用され，バスコントロールの切換えのための完全なプロトコール機能を提供する.このリクエスト/グラントのシーケンスはHOLD/HOLDAの動作に類似しているが，一つのピンが両方の働きをする点が特に異なる.

まずリクエストパルスが$\overline{RQ}/\overline{GT}$ピンに到達すると，CPUはアドレスバス，データバス，およびコントロール信号をフロートの状態にする.リクエストを出した装置は，クロックの次のLからHへの遷移のところでCPUからのグラントパルスを受け取るとバスの使用が可能になる.最後にバスの使用が終了してCPUにバスのコントロール権を返すには，今までのバスマスタがバスコントロール権を解放し，同じ$\overline{RQ}/\overline{GT}$線にリリースパルスを出すことでその動作を完了する.

8086は偶数アドレスバイトおよび奇数アドレスバイトを含んでいる二つの別々のバンクで物理的には構成されているので，8ビット構成のDMAコントローラがメモリ中で論理的に連続しているバイトをアクセスするためには，これらのバンクを交互に選択するような回路を構成しなければならない.

8089 I/Oプロセッサは，高速の8ビット装置を8086ベースシステムにインタフェースするのを容易にする(12·2節参照).この8089はデータ処理機能をもった二つのDMAチャネルと，I/O動作用の特別の命令セットを持っており8ビット/16ビットの周辺装置を16ビット/8ビットのバスに適合させたり，メモリ・メモリ間，I/O-I/O間のデータの転送等も可能にする(12章参照).

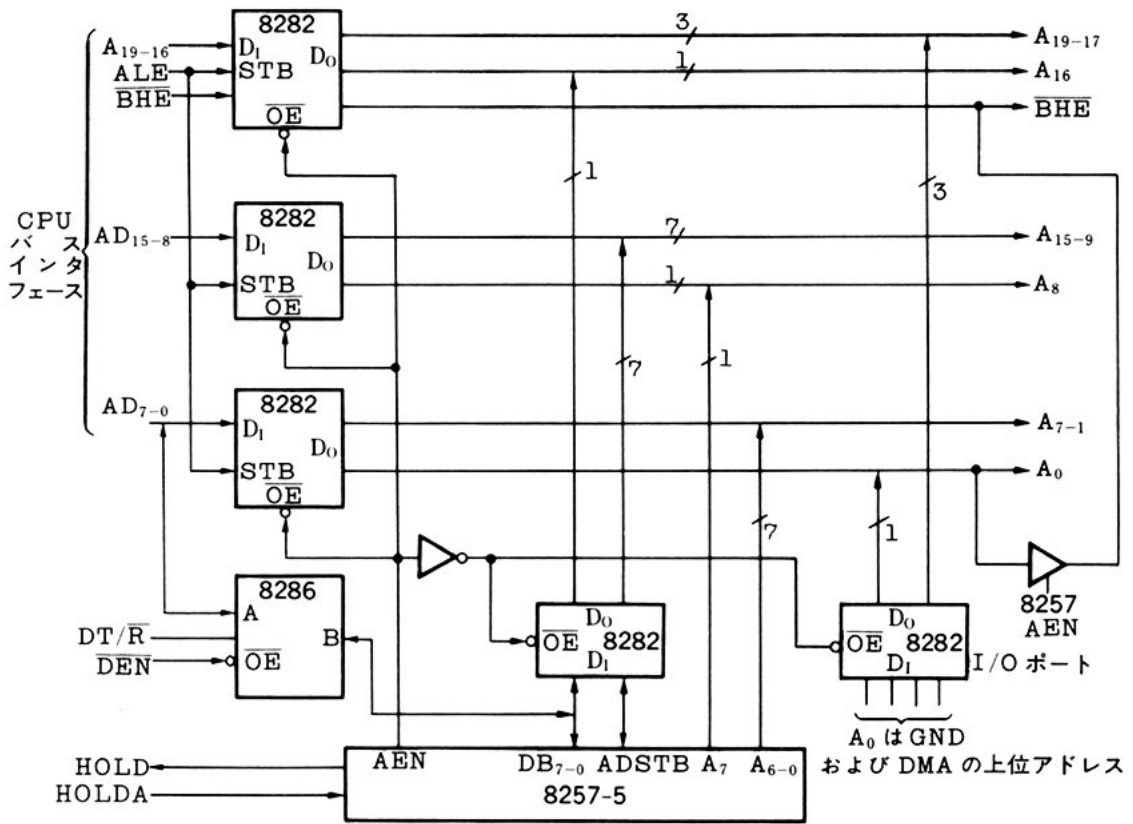



図 4・3 DMA コントローラの使用

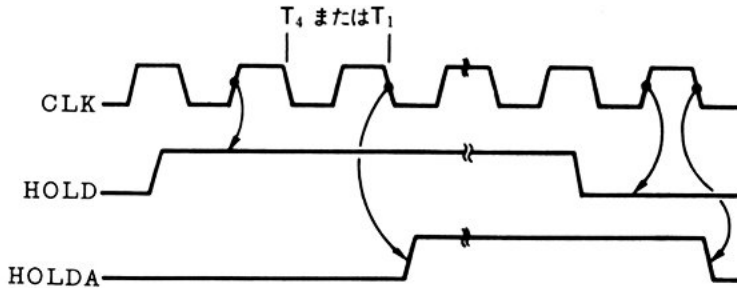



図 4・4 HOLD/HOLDA タイミング（ミニマムモード）

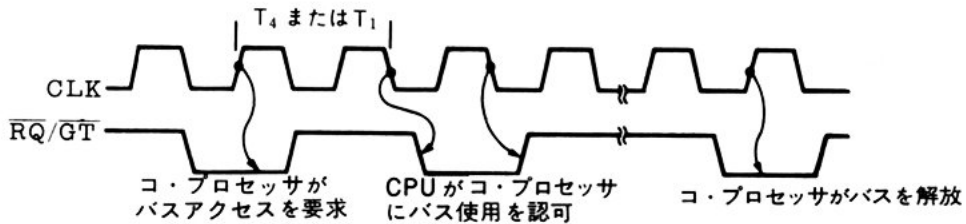



図 4・5 RQ/GT タイミング（マキシマムモード）

8086/8088 に I/O 装置を接続する場合には，次のような注意が必要である．

8086 の場合：

16 ビットの I/O は，1 ワードのデータを 1 回のバスサイクルで転送するためには，偶数アドレスに接続しなければならない．これはメモリアクセスの場合と同じである（p. 28 参照）．

8 ビットの装置を接続する場合は，偶数または奇数のどちらのアドレスに接続してもかまわないが，その装置内のレジスタのアドレスは，すべて偶数か，またはすべて奇数にしなければならない．

8088 の場合：

8088 では 1 バスサイクル当り 1 バイトの転送を行うので，これに 16 ビットの装置を接続する場合は，1 ワードのデータの転送は，1 度に 8 ビットずつ，2 バスサイクルで行う．8088 は従来の 8 ビット CPU と同様に考えてよいが，8086 とのプログラムの互換性を保つためには，前記同様レジスタのアドレスはすべて偶数，またはすべて奇数にすべきである．

5. プロセッサ動作のコントロール

CPU 動作の開始手順について述べ，プロセッサの状態情報についてまとめて説明する．8086で新しく導入された命令キューの動作は命令の実行サイクルと対比して述べている．次に割込みは8259Aと組み合わせたベクタ割込みのほかソフトウェア割込みなど各種割込みについても記述している．

5・1 CPUのリセットからスタートアップへ

8086/8088がリセットにより初期化されると，各セグメントレジスタおよびインストラクションポインタ（IP）は，それぞれ次のように初期化される．

IP（インストラクションポインタ）：0000H

CS（コードセグメント）レジスタ：FFFFH

DS（データセグメント）レジスタ：0000H

SS（スタックセグメント）レジスタ：0000H

ES（エックストラセグメント）レジスタ：0000H

フラグ：クリヤされる．

キュー：空になる．

コードセグメントレジスタはFFFFHに，そしてインストラクションポインタ（IP）は0に初期化されるので，リセットの後に最初に実行される命令は，メモリロケーションFFFF0Hからとなる．ここには通常インタセグメント（セグメント外）の直接ジャンプ命令が置かれ，その跳び先はシステムプログラムの始まりの点になっている．LOC-86でロケートするときに，この開始アドレスを指定すれば，その最初の命令を指定するJMP命令を自動的にその場所に挿入してくれる．リセット信号はアクティブHIGHの信号で，8284Aクロック発振器を通して供給される．リセット後のアドレスバスおよび各コントロールバスの状態は，表5・1のようになる．

リセットパルスの幅は，電源投入時は最小50μs，その他の場合はCPUの4クロック周期分を必要とする．CPU内部のリセット信号はクロックパルスに同期しており，図5・1に示すように，クロックの立上りで外部リセット入力を確認のうえ内部リセットをセットし，次のクロックがLOWの区間バスは“1”となり，次のクロックの立上りで3ステートとなる．

CPUのコマンドおよびバスコントロール線は，アクティブでない場合の信号のレベルが，HIGHレベルの規格値以下にならないように，22kΩ程度のプルアップ抵抗を挿入しなければならない．

表 5・1　リセット後の各信号の状態

信号名	状態
AD_{15-0}	3ステート
A_{19-16}/S_{6-3}	〃
BHE/S_7	〃
$\overline{S_2}/(M/\overline{IO})$	"1" になり，その後3ステート
$\overline{S_1}/(DT/\overline{R})$	〃
$\overline{S_0}/\overline{DEN}$	〃
$LOCK/\overline{WR}$	〃
$\overline{RD}$	〃
$\overline{INTA}$	〃
ALE	0
HOLDA	0
$\overline{RQ}/\overline{GT_0}$	1
$\overline{RQ}/\overline{GT_1}$	1
QS_0	0
QS_1	0

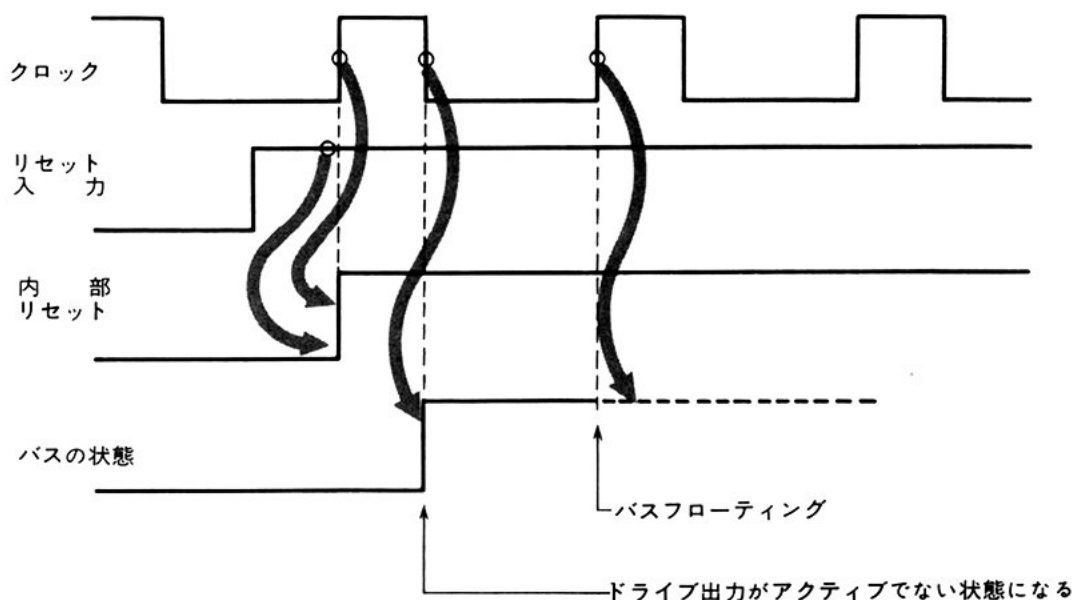



図 5・1　リセット時のバスの状態

5•2 命令キューとキューステイタス

CPUの実行ユニット(EU)がバスインタフェースユニット(BIU)の使用を伴わない命令を実行中は，BIUは空いているので，先行してメモリからその後に続く命令をフェッチすることができる．この先行してフェッチした命令は**命令キュー**と呼ばれるCPU内部の**FIFO RAM**に格納される．8088では4バイトまで，8086では6命令バイトまで可能で，EUはこのキューから命令をフェッチすることによりBIUを専有することを防いでいる．

8088のBIUは，キューの中に1バイトの空きができると，次の命令をフェッチしてくるので，EUが命令のフェッチのために直接BIUに要求を出すことはない．

8086の場合は，キュー中に2バイトの空きができるまで命令のフェッチを行わないということ以外は8088と同じで，この場合には通常1回のフェッチで2バイト(1ワード)を獲得する．

プログラムが奇数アドレスからフェッチをさせようとした場合は，BIUは自動的に奇数アドレスから1バイト読み，そしてそれからその後に続く偶数アドレスから2バイト(ワード)のフェッチをする．

命令が連続的に行われているときは，キューに含まれている命令は現在実行しているものの次に実行する命令になっているが，もし実行ユニットがコントロールを他のロケーションに移す命令(CALL, JMPなど)を実行すると，BIUは今までのキューをリセットし，新しいアドレスから命令をフェッチしてきて，それを直ちにEUに送る．その後キューは，また新しいロケーションから再び満たされる．

EUがメモリまたはI/Oのリード/ライトを伴う命令を実行する場合には，BIUはそのための使用に解放され，命令のフェッチは中断される．

このキューの状態を外部に知らせる信号としてQS_0，およびQS_1がマキシマムモードの場合に出力され，コ・プロセッサ(8087)による外部命令セットの拡張や，インサーキットエミュレータ(ICE)モジュールによるトレース動作等に使用される．

EU：JMP 命令の場合は，命令実行中にキューを再初期化．
BIU：キューは空のため，BIU は1バスサイクルで二つのオブジェクトコードバイトをフェッチ．
キュー＝4バイト

EU は，キューから最初の2バイト（命令コード）をフェッチし，それぞれの命令に必要なクロックサイクルで，その命令の実行を行う．

BIU は2オブジェクトコードバイトをフェッチ．
キュー＝4バイト

BIU はさらに二つのオブジェクトコードバイトをフェッチ
キュー＝6バイト　（満杯）

BIU は，EU が命令の実行を完了するまで，適当なクロックサイクル分，アイドル状態になる．

EU はキューから次のオブジェクトコードバイトをフェッチし，次の命令の実行を始める．
BIU は二つのオブジェクトコードバイトをフェッチしキューの再補充をする．
キュー＝6バイト　（満杯）

図 5・2　命令の実行とキューの状態

表 5・2　キューの状態

QS_1	QS_0	キューの状態
0	0	動作なし：前のクロックサイクルの間にキューから何も持ってこられなかった．
0	1	最初のバイト：キューから持ってこられたバイトが命令の最初のバイトであった．
1	0	キューが空き：コントロール移行命令の実行結果として，キューが再初期化された．
1	1	続きバイト：キューから持ってこられたバイトが命令の続きのバイトであった．

5・3 状態情報ライン

アドレスラインA_{19}～A_{16}は，T_1サイクルの間はメモリ動作のための最上位の4アドレス信号を出力するが，次のT_2, T_3, T_WおよびT_4の間には状態情報S_6～S_3をそれぞれ同じライン上に出力する．同様にバスハイイネーブル($\overline{BHE}$)もS_7とマルチプレクスされて出力される．これらの状態情報の意味を表5・3に示す．

S_7～S_3まではミニマム/マキシマム構成のいずれに対しても出力されるが，$\overline{S_2}$, $\overline{S_1}$および$\overline{S_0}$はマキシマムモードの場合に，それぞれミニマムモードのIO/$\overline{M}$, DT/$\overline{R}$および$\overline{DEN}$の代わりに出力されるもので，この信号を8288バスコントローラが受け取り，それをデコードすることにより必要なコントロール信号を発生する（表5・3参照）．

8288を使用する場合はメモリライト/IOライト信号は，通常のタイミングで出力されるコントロール信号に加えて，それらの信号よりも1クロックサイクル先行した$\overline{AMWC}$（アドバンストメモリライトコントロール）および$\overline{AIOWC}$（アドバンストI/Oライトコントロール）が同時に得られ，ライトアクセスのタイミングの改善に役立っている．

S_3およびS_4は，その命令の現在のデータアクセスに，ES, SS, CS，またはDSレジスタのうちのどのセグメントレジスタが使用されているかを示している．

S_5は内部の割込みイネーブルフラグの状態を示しており，クロックサイクルの初めで毎回更新される．

S_6はつねに0で，8086がバスに接続されていることを示し，S_7は予備で，現在は使用されていない．

ステイタス情報の出力のタイミングは，その命令の前のバスサイクル（この種の命令では通常，2バスサイクル以上を要する）のT_4ステートのクロックの立上りのエッジで，CPUは状態情報ラインにこれらの信号を出力し，8288は各クロックサイクルの立下りでその情報をサンプルし，デコードする．次に，8288はALEを出力して次のバスサイクルをスタートする．最後に，T_3ステートで状態情報ラインがパッシブの状態になると，8288はそのサイクルを終了する．

表 5・3 状態情報の意味

ステイタス情報			意味	
S_7			予 備	
S_6			つねに0	
S_5			割込みイネーブルフラグの状態	
S_4	S_3		セグメントレジスタ状態情報	
0	0		ES	
0	1		SS	
1	0		CS	
1	1		DS	
$\overline{S_2}$	$\overline{S_1}$	$\overline{S_0}$	バスサイクルのタイプ	8288 信号
0	0	0	割込みのアクノレージ	$\overline{INTA}$
0	0	1	READ I/O	$\overline{IORC}$
0	1	0	WRITE I/O	$\overline{IOWC}$, $\overline{AIOWC}$
0	1	1	HALT	な し
1	0	0	命令フェッチ	$\overline{MRDC}$
1	0	1	READ メモリ	$\overline{MRDC}$
1	1	0	WRITE メモリ	$\overline{MWTC}$, $\overline{AMWC}$
1	1	1	パッシブ	な し

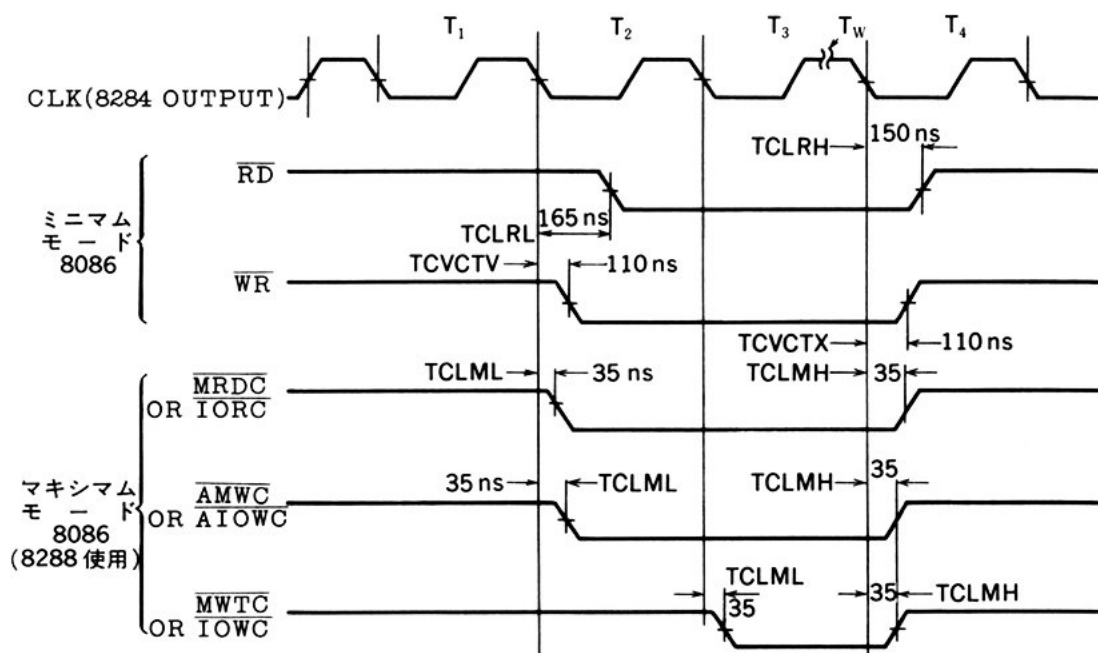



図 5・3 コントロール信号のタイミング

5・4 割込みポインタテーブル

8086/8088の割込み端子はハード的にはNMI（ノンマスカブルインタラプト）とINTRの2端子であるが，あらかじめ定められた割込みおよびユーザ定義のソフトウェア割込み（割込みコントローラ8259Aと組合せ）があり，0～255種類の割込みが可能になっている．割込みの優先度は，あらかじめ定められた割込みでは前もって決まっており，8259Aを使う場合は，その接続順序により決定される．

割込みポインタテーブルは，システムメモリの0～3FFH（1Kバイト）が割り当てられており，2ワード（4バイト）をペアとして，256種類の割込みポインタを格納できる．この2ワードのうちの下位アドレスにはIPオフセットが，上位アドレスにはCSベースアドレスが格納されており，2・3節で述べたのと同じ方法で1Mバイトまでのどこにある割込み処理ルーチンへもジャンプ可能である．

0～13Hまではあらかじめ定められた割込みのポインタ，14H～7FHはインテル社で予約している領域で，将来インテルから供給されるハード/ソフトとの互換性を保つ必要がある場合は使用を避けたほうがよい．

80H～3FFHがユーザに解放されている領域である．各ポインタは4バイト跳びで格納されているが，CPUは命令または8259Aによって供給される割込みタイプナンバに4を掛ける（左へ2ビットシフト）ことにより，これらの指定を行う．

割込みが呼び出されると，CPUは特定のタイプを持ったベクタによって指定されたロケーションにコントロールを渡す．ユーザはそのロケーションにその割込みのための処理ルーチンを置き，割込みベクタテーブルを，そのサービスルーチンのアドレスで，前もってプログラムの最初の部分で初期化しておかなければならない．

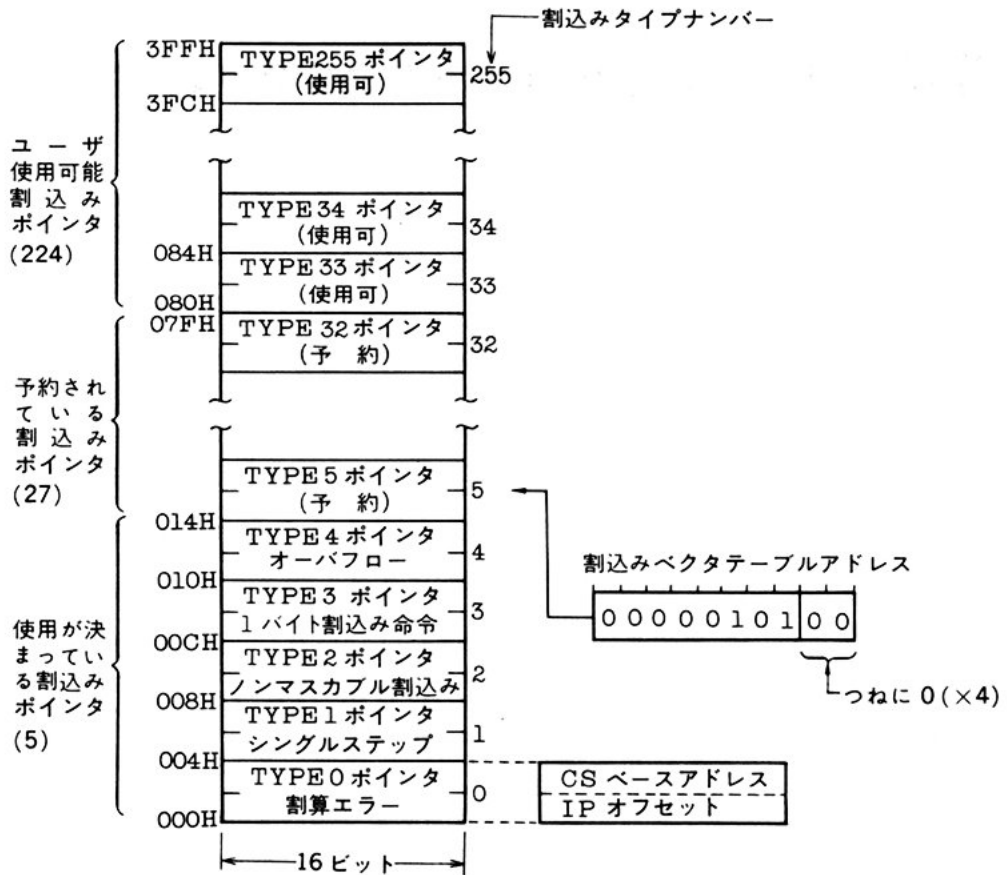



図 5・4 割込みポインタテーブル

```
INT_POINTERS        SEGMENT
; INTERRUPT POINTER TABLE. LOCATE AT 0H, ROM-BASED
TYPE_0      DD      ?               ; DIVIDE-ERROR NOT SUPPLIED IN EXAMPLE
TYPE_1      DD      ?               ; SINGLE-STEP NOT SUPPLIED IN EXAMPLE.
TYPE_2      DD      POWER_FAIL      ; NON-MASKABLE INTERRUPT
TYPE_3      DD      ?               ; BREAKPOINT NOT SUPPLIED IN EXAMPLE.
TYPE_4      DD      ?               ; OVERFLOW NOT SUPPLIED IN EXAMPLE.
;SKIP RESERVED PART OF EXAMPLE
            ORG     32*4            ;
TYPE_32     DD      ?               ;8259A IR0-AVAILABLE
TYPE_33     DD      ?               ;8259A IR1-AVAILABLE
TYPE_34     DD      ?               ;8259A IR2-AVAILABLE
TYPE_35     DD      TIMER_PULSE     ;8259A IR3
TYPE_36     DD      ?               ;8259A IR4-AVAILABLE
TYPE_37     DD      ?               ;8259A IR5-AVAILABLE
TYPE_38     DD      ?               ;8259A IR6-AVAILABLE
TYPE_39     DD      ?               ;8259A IR7-AVAILABLE
;
;POINTER FOR TYPE 40 SUPPLIED BY PL/M-86 COMPILER
;
INT_POINTERS        ENDS
```

図 5・5 割込みポインタテーブルのセットアッププログラム例

5・5 割込みの種類（あらかじめ定義された割込み）

あらかじめ定義された割込みに属するものは，割込みポインタテーブルの最初の五つの割込みで，優先度の最も高いものである．

（a） **タイプ0割込み（割算エラー）**　このタイプの割込みは，割算動作による商が最大値を超えるときに出される．0で割る場合がこれに相当する．この割込みはノンマスカブルで，割算命令の一部として実行される．

（b） **タイプ1割込み（シングルステップ）**　このタイプの割込みは，フラグレジスタ中でTF（トラップフラグ）がセットされた1命令後に起こる．これはソフトウェアによるシングルステップを可能にするもので，この割込みルーチンはシングルステップのためのルーチンでなければならない．割込みシーケンスとしては，フラグおよびプログラムカウンタの退避後，TFフラグをリセットし，シングルステップルーチンが正常に実行されるのを可能にする．テスト中のルーチンに戻るには割込みからの復帰により，IP, CSおよびTFを回復し，シングルステップルーチンに戻る前に，テスト中のプログラムの次の命令の実行を可能にする．シングルステップはフラグレジスタ中のIFビットでマスクされない．

（c） **NMI（ノンマスカブル割込み）**　この割込みは，優先度の最も高い割込みで，マスクはできない．この割込み入力はエッジトリガであるが，CPUクロックに同期しており，認識されるためにはCPUの2クロックサイクルの間HIGHでなければならない．また逆にLOWの期間も最小2CPUクロック分なければならない．この割込みは通常，電源異常などの緊急割込みに使用される．

（d） **1バイト割込み**　この割込みは，単一バイトのソフトウェア割込みの特別な形で，おもにソフトウェアデバックのためのブレークポイント割込みとして使用される．この割込みはマスクできない．

（e） **オーバフロー割込み**　この割込みは，オーバフローフラグ（OF）がフラグレジスタ中でセットされ，そしてINTO命令が実行されたときに発生する．この命令はオーバフローエラーのサービスルーチンへの分岐を可能にする．また，これもマスクできない割込みである．

5・6 その他の割込みと割込みシーケンス

〔1〕 **ユーザ定義のソフトウェア割込み**　2バイトの割込み命令INTnnで，ソフトウェアの割込みを発生することができる．この命令の最初のバイトはINTのオペコードで，2番目のバイト（nn）は実行する割込みのタイプナンバを含んでいる．INT命令は割込みイネーブルフラグによりマスクはされない．この命令は，そのメモリ中のロケーションがコーリングプログラムにはわからないダイナミックリロケータブルなルーチンへポインタテーブル経由でコントロールを渡すのに使用可能である．これらのものは割込みアクノレージのバスサイクルは行わず，IFおよびTFフラグをリセットすることにより，その後のマスカブル割込みを禁止する．これらの割込みタイプに対するベクタは，命令中に包含されるか，あるいは指定されるかのいずれかである．

〔2〕 **ユーザ定義のハードウェア割込み**　マスカブル割込みは8086/8088 INTR端子で活性化，ステイタスレジスタのIFビットでマスクされ，各命令の最後のクロックサイクルの間にチェックされる．割込み受付けが保証されるには，CPUから割込みアクノレージが出るまでINTR端子をHIGHにする．

図5・7に割込みアクノレージシーケンスを示す．これはINTR端子からの割込みに対してだけ発生されるもので，二つの$\overline{\mathrm{INTA}}$バスサイクルから成っており，最初の$\overline{\mathrm{INTA}}$バスサイクルは割込みアクノレージサイクルが進行中であることを知らせ，次の$\overline{\mathrm{INTA}}$サイクルでシステムが割込みタイプナンバを供給する準備をすることを可能にする．CPUは最初のバスサイクルでは，バス上の情報は受け取らず，2回目のサイクルでデータバスの下位8ビット上のタイプナンバ情報を受け取る．すなわち，割込みタイプナンバを供給する機器は8086の16ビットバスの下位8ビットに接続されていなければならない．8086の割込みアクノレージシーケンスは，8080/85のようなリスタートあるいはコール命令によりコントロールを受け渡す方法とは異なり，そのシーケンスの一部として命令の発生（CALL等）は行わない．マキシマムモードの場合，$\overline{\mathrm{LOCK}}$信号が最初のサイクルのT_2から，2番目のT_2まで出力され，$\overline{\mathrm{INTA}}$サイクル中のHOLDを禁止している．

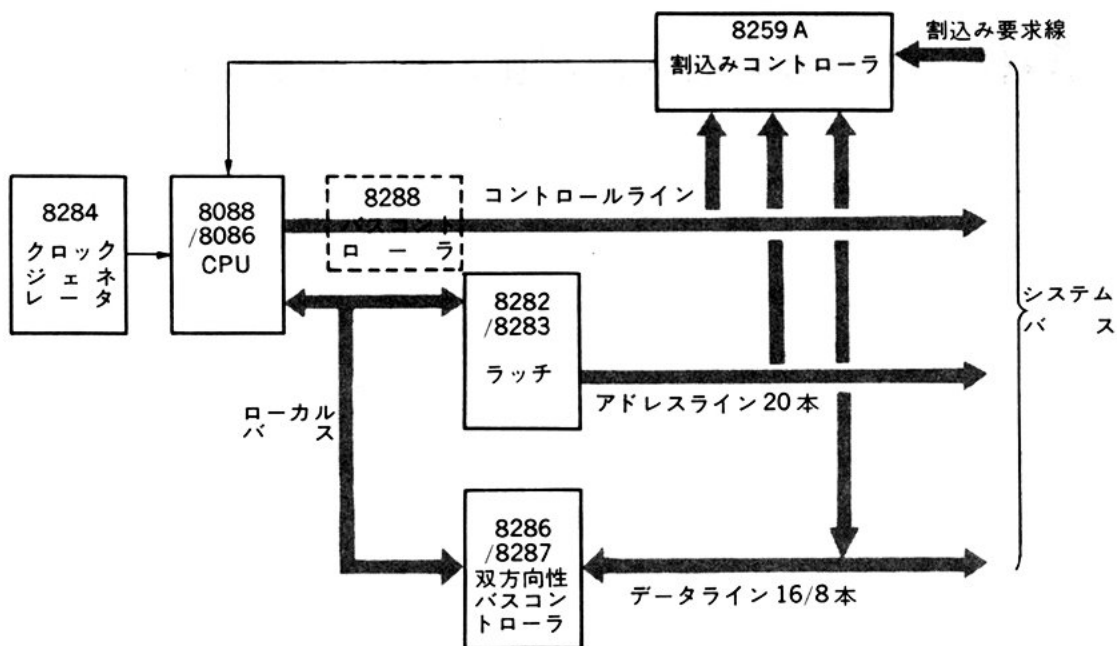



図 5・6　ミニマムモード/マキシマムモードのバス構成と 8259A の使用

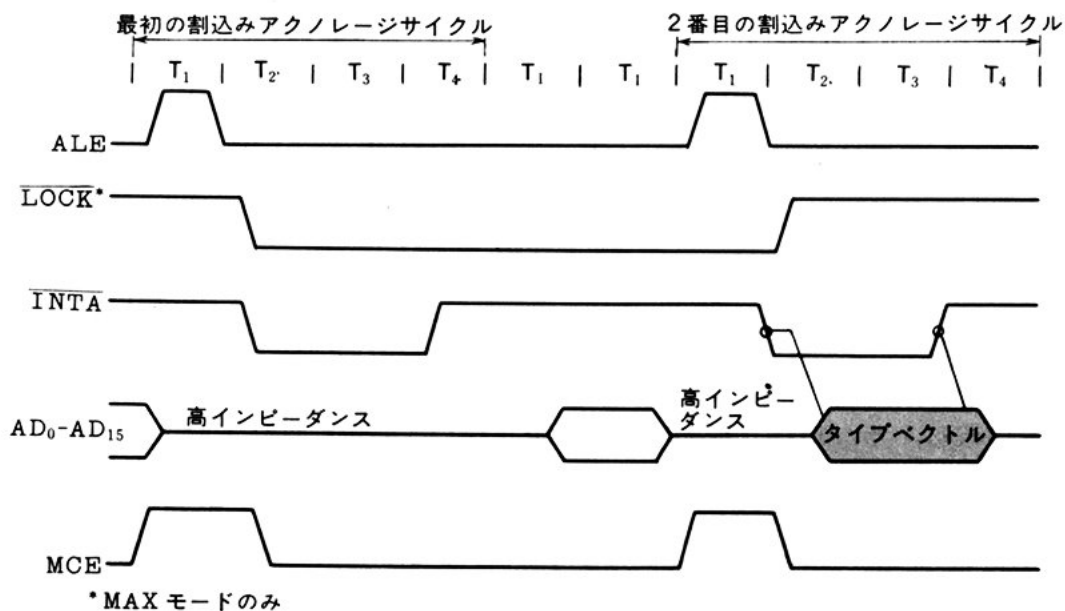



図 5・7　割込みアクノレージシーケンスと 8259A が CAS アドレスを
8086 のローカルバスにのせる MCE タイミング

6. 命令セット

8086の全命令セットを機能別に6個のグループに分け，その動作を図解する．7章のアドレッシングモードとの関連，およびレジスタとの組合せにより多彩な命令のバリエーションが可能である．また，新しく加わったストリング機能は，それらの命令をより強力なものにしている．

6・1 命令のエンコーディング

機械語命令は1バイトのものから6バイトのものまであるが，その中で最も重要な部分は最初の2バイトである．その命令のフォーマットを図6・1に示す．

命令の最初の6ビットは一般的に命令の基本的なタイプを表すOPコードで，**D領域**は後に続くオペランドの方向を示している．たとえば“1”の場合，2番目のバイト中の“REG”領域がディスティネーション（宛先）オペランドであることを示し，“0”の場合は，それがソースオペランドであることを示す．**W領域**はバイトまたはワード動作の区別で，0の場合にバイト，1の場合にワードを表す．

いくつかの命令中の最初のバイト中にはさらに三つのビット領域，S, V, Z（巻末の付録3参照）が存在し，SはWとともに使用され，算述命令中のイメディエートデータの符号付き拡張（W＝1の場合，16ビットデータ）を示す．Vは，シフトまたはローテートの数が1かあるいは可変（C_L レジスタ中で指定）であるかを指定する．**Z**は条件付きのループおよび繰返し命令中での比較ビットのゼロフラグの状態を表し，1の場合はゼロフラグがセットのときに，0の場合はリセットの場合に，そのループ/繰返し動作を行うことを示す．

命令の第2番目のバイトは通常，その命令のオペランドを指定し，**MOD領域**はそのオペランドのうちの一つがメモリ中にあるものかどうか，両方のオペランドがレジスタであるかどうか，等を指定する．**REG領域**は命令オペランドのうちの一方がレジスタであることを示し，メモリに対するイメディエートの場合はその動作のタイプを指定するためのOPコードの拡張として使用される．

R/M（レジスタ/メモリ）**領域**はモード領域のセットの状態に依存し，MODが11（レジスタ・レジスタモード）の場合は，R/Mは2番目のレジスタオペランドを指定する．MODがメモリモードになっている場合は，R/Mはどのようにしてそのメモリオペランドの実効アドレスを算出するかを指定する．

命令の第3番目から第6番目までのバイトはオプションで，通常メモリオペランドのディスプレースメント値または，イメディエート定数オペランドの実際の値を含んでいる．以上の動作をまとめたものを表6・1に示す．

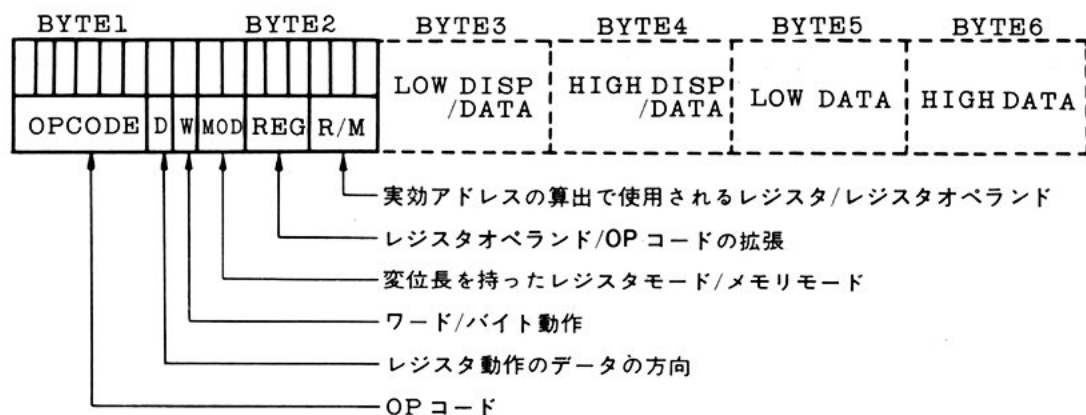



図 6・1 8086/8088機械語フォーマット

表 6・1 命令コードのエンコーディング

R/M	メモリモード MOD=00 (ディスプレースメントなし)	MOD=01 (8ビットディスプレースメント)	MOD=10 (16ビットディスプレースメント)	レジスタモード MOD=11 (ディスプレースメントなし) W=0	W=1
000	(BX)+(SI)	(BX)+(SI)+D8	(BX)+(SI)+D16	AL	AX
001	(BX)+(DI)	(BX)+(DI)+D8	(BX)+(DI)+D16	CL	CX
010	(BP)+(SI)	(BP)+(SI)+D8	(BP)+(SI)+D16	DL	DX
011	(BP)+(DI)	(BP)+(DI)+D8	(BP)+(DI)+D16	BL	BX
100	(SI)	(SI)+D8	(SI)+D16	AH	SP
101	(DI)	(DI)+D8	(DI)+D16	CH	BP
110	直接アドレス	(BP)+D8	(BP)+D16	DH	SI
111	(BX)	(BX)+D8	(BX)+D16	BH	DI

6・2 データ転送命令

メモリ・レジスタ間と同じように，ALまたはAXレジスタとI/Oポートの間でバイトおよびワードデータを移動する命令も含めて14種のデータ転送命令がある．また，スタック操作命令もこのグループ中に含まれる．

〔1〕 **汎用のデータ転送**

MOVディスティネーション，ソース：MOVは，バイトまたはワードデータを，ソースオペランドのアドレスからディスティネーションオペランドのアドレスに転送する．

PUSHソース：PUSHは，スタックポインタSPを2減じ，それからそのソースオペランドからのワードをSPによって新しく指定されたスタックのトップに転送する．PUSHは，サブルーチンコールや割込みの場合にレジスタ値やフラグ等の退避に使用する．

POPディスティネーション：POPは，現在SPによって指定されているスタックのトップにあるワードを，そのディスティネーションオペランドに転送し，それから2だけSPを増加し，新しいスタックを指示する．POPは，サブルーチンや割込み処理プログラムからの復帰などの場合に，スタック中のデータを元のレジスタやメモリに返すのに使用する．

XCHGディスティネーション，ソース（交換）：XCHGは，ソースとディスティネーションオペランドの内容を交換する．

XLAT翻訳テーブル（翻訳）：XLATは，ALレジスタ中のバイトデータを，ユーザの供給する256バイトの翻訳テーブルからのバイトと置き換える．この場合のベースレジスタはBXが使用され，ALはそのテーブルに対するインデックスとしての働きをし，その内容の値に対応するテーブル中のオフセットのところのバイトと置き換えられる．その動作を図6・6に示す．

XLATはASCIIとEBCDICコードの相互変換などの，あるコードから他のコードへの変換などに便利に使用できる．

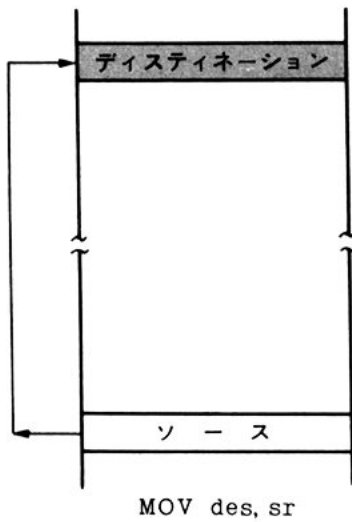



図 6・2 データ転送命令（1）

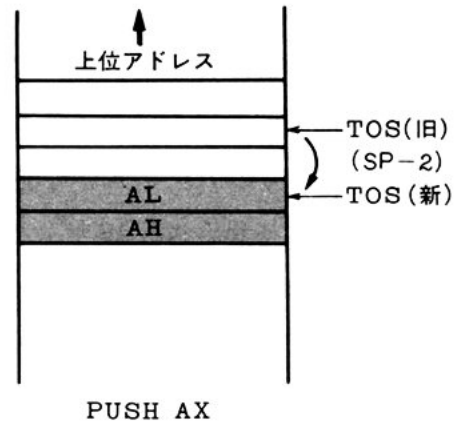



図 6・3 データ転送命令（2）

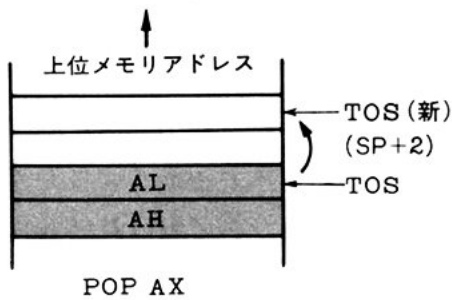



図 6・4 データ転送命令（3）

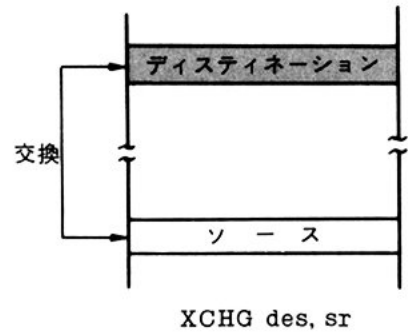



図 6・5 データ転送命令（4）

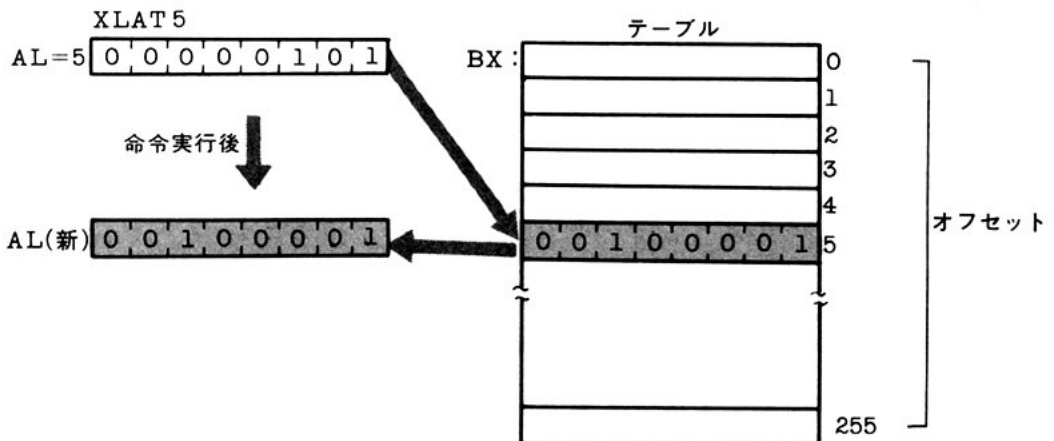



図 6・6 XLAT（翻訳）の動作

Mnemonics © Intel, 1978

〔2〕 **入出力命令**

IN アキュムレータ, ポート番号：IN 命令は，指定された入力ポートからバイトまたはワードデータを，AL または AX レジスタへそれぞれ転送する．ポート番号の指定は二つの方法が可能で，直接のバイト定数として指定する場合は 0～255 ポートまでのアクセスが可能で，DX レジスタ中にあらかじめ設定した数値で間接指定する場合は，16 ビットあることから，0～65 535 までの I/O ポートの指定ができる．

OUT ポート番号, アキュムレータ：OUT は，AL または AX レジスタから指定された出力ポートに，バイトまたはワードデータを転送する．ポート番号の指定は IN の場合と全く同じである．

〔3〕 **アドレスオブジェクトの転送**　　この命令は定数や変数の値ではなく，変数のアドレスを操作するもので，リスト処理，ベースをもった変数，およびストリング動作等の場合のベースアドレスの設定に使用される．

LEA ディスティネーション, ソース（実効アドレスのロード）：LEA は，ソースオペランドのオフセット（その値ではない）をディスティネーションオフセットに転送する．このソースオペランドはメモリオペランドでなければならず，ディスティネーションオペランドは，16 ビットの汎用レジスタでなければならない．たとえば，XLAT 命令で使われる翻訳テーブルのアドレスを BX レジスタにロードする場合などに使用される．

LDS ディスティネーション, ソース（DS を使ったポインタのロード）：LDS は，ソースオペランドからの 32 ビットのポインタ変数をディスティネーションオペランドおよび DS レジスタに転送する．すなわち，ポインタのオフセットワードは，ディスティネーションオペランドとして指定可能な任意の 16 ビット汎用レジスタに転送され，そのポインタのセグメントワードは DS レジスタに転送される．これはストリング命令の場合の DS, SI の初期設定等に使用される．

LES ディスティネーション, ソース（ES を使ったポインタのロード）：LES は，DS の代わりに ES を使用するということ以外，動作としては LDS と同じである．このディスティネーションオペランドとして DI を指定することにより，ストリング動作の場合の ES, DI の初期設定に使用される．

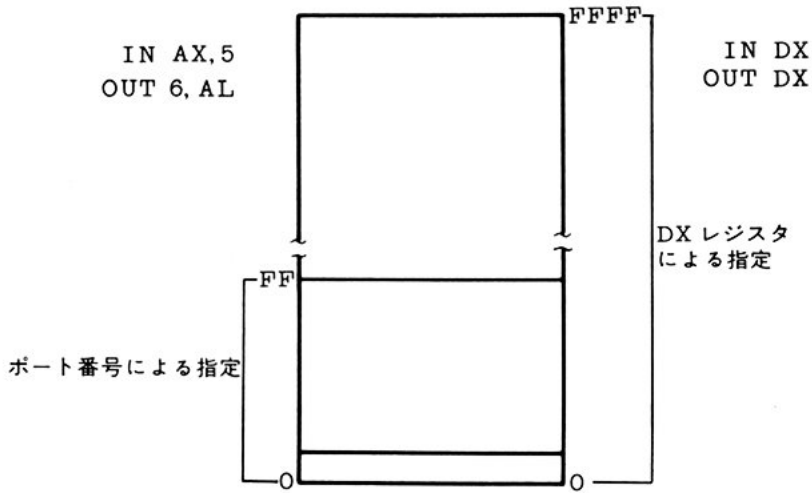



図 6・7 入出力命令による I/O ポート指定

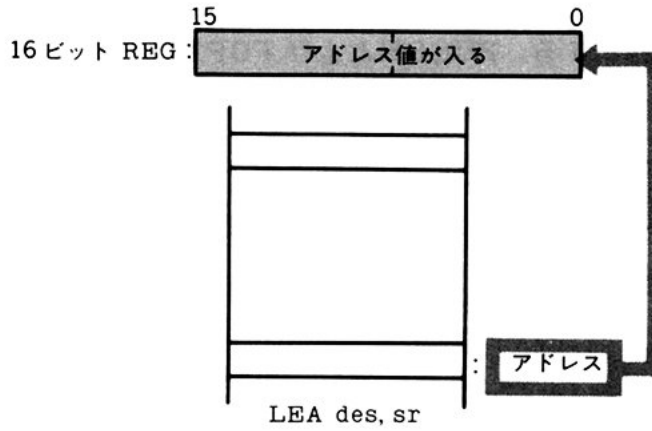



図 6・8 アドレスの転送（1）

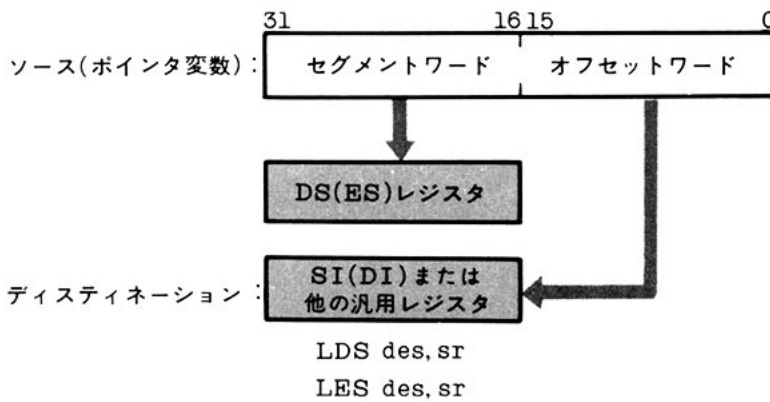



図 6・9 アドレスポインタのロード

［4］ **フラグ転送命令**

LAHF（フラグからAHレジスタへロード）：LAHFは，SF，ZF，AF，PFおよびCFなどのフラグをAHレジスタにコピーする．この命令は，8080/85のプログラムが8086/88で走るように変換するために用意されたものである．

SAHF（フラグにAHレジスタを格納）：SAHFは，AHレジスタの内容をSF，ZF，AF，PFおよびCFビットに格納する．OF，DF，IFおよびTFは影響されない．

PUSHF：PUSHFは，SPを2減じ，それからすべてのフラグを，SPによって指定されるスタックのトップにあるワード領域に転送する．

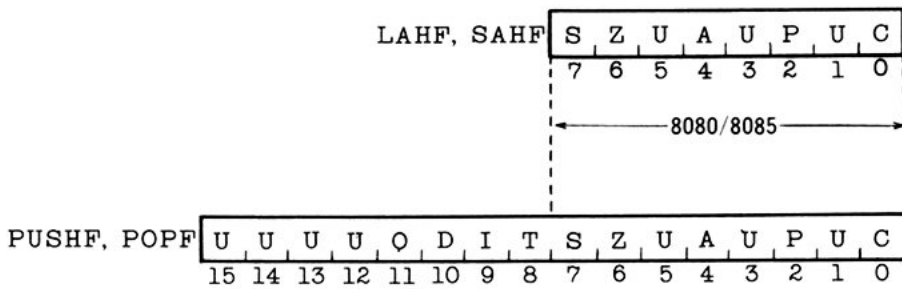
POPF：POPFは，SPによって指定された現在スタックのトップにあるワードを8086/88のフラグ中に転送する．それからSPは2だけ増加され，スタックの新しいトップを指示する．PUSHFおよびPOPFはサブルーチンや割込みなどのコーリングプログラムのフラグの退避や復帰に使用される．その他，シングルステップの場合のTFフラグなどのセッティングをプログラムにより変更したりするときに，そのメモリの相当するビットを変更した後，そのフラグをPOPすることにより達成できる．

U：未定義
O：オーバフローフラグ
D：ディレクションフラグ
I：割込みイネーブルフラグ
T：トラップフラグ
S：サインフラグ
Z：ゼロフラグ
A：補助キャリーフラグ
P：パリティフラグ
C：キャリーフラグ

図 6・10 フラグの構成

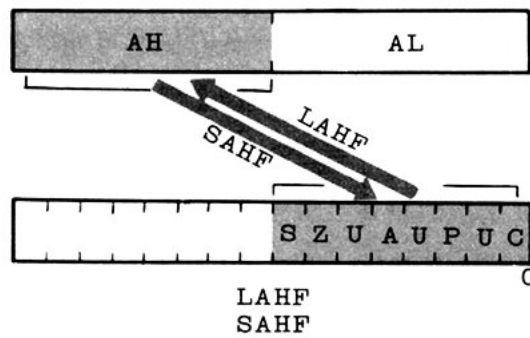



図 6・11 フラグ転送命令（1）

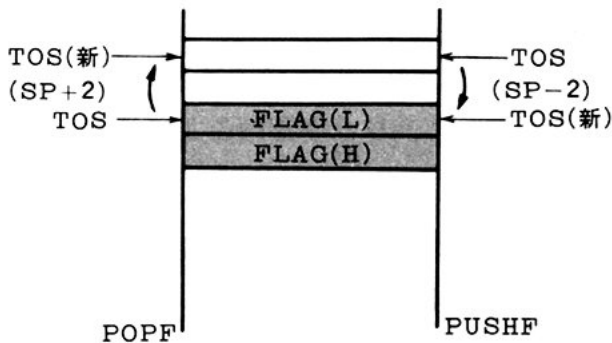



図 6・12 フラグ転送命令（2）

6・3 演算命令

［1］ **数値の表現法** 8086/88で扱う数値には，表6・2に示すように四つのタイプがある．2進数は8または16ビット長が可能で，10進数は**パック***された10進数に対しては1バイト当り2ディジット，そして**アンパック***の10進数は1バイトに1ディジット格納される．符号なし2進数は8または16ビット長が可能で，8ビットの場合は0～255まで，16ビットの場合は65,535までの範囲が可能である．符号付き2進数は8または16ビット長が可能で，そのMSBはその数の符号を表し，0が正，1が負となる．負数は2の補数表現で，8ビットの場合は－128から＋127，16ビットの場合は－32,768から＋32,767の範囲になる．

パックされた10進数は符号なしのバイト値として格納されており，その各**ニブル**が一つの10進数を表し，それぞれ0～9の値が可能であることから，その範囲は0～99までになる．この場合の加減算は2段階で行われ，まず符号なしの2進演算命令が使用され，ALレジスタにその中間結果を入れ，それからその値を最終的なパックされた10進数の結果に調整（DAA，後述）する動作が行われる．

アンパックの10進数は符号なしのバイト値として格納されており，その数の大きさは下位ニブルにより表されるので，0～9までの数が可能であり，その上位ニブルは0でなければならない．このアンパックの10進数の演算も2段階で行われ，まず符号なしの四則演算の中間結果がALレジスタに作成され，それを最終的なアンパックの10進数に調整する（AAA）ための動作が行われる．

このアンパックの数値表現は，その上位ニブルが0であること以外はASCII（この場合は3）の数値表現に類似しており，下記のことに注意することによりASCII表現の数の演算が可能になる．

・演算命令の実行前に，ASCII数の上位ニブルの3を0にセットする．

・アンパックの10進演算は，その結果の上位ニブルは0として返すので，それを有効なASCII数に変換するためには，それを3にセットしなければならない．

---

* 10進数等を表すのに1バイトに2ディジット分詰め込んだ場合をパックされているといい，1バイトに1ディジット入れ，上位ニブルに0を詰め込んだ場合をアンパックという．

表 6・2 8ビット数の数値表現

16進表現	ビットパターン	符号なし 2進数	符号付き 2進数	アンパックの10進数	パックの10進数
07	00000111	7	+7	7	7
89	10001001	137	−119	無効	89
C5	11000101	197	−59	無効	無効

```
;SUM THE CONTENTS OF TABLE INTO AX
TABLE     DW      50 DUP(?)
;NOTE SAME INSTRUCTIONS WOULD WORK FOR
;TABLE    DB      25 DUP(?)
;TABLE    DW     118 DUP(?), ETC.

          SUB     AX, AX            ;CLEAR SUM
          MOV     CX, LENGTH TABLE;LOOP TERMINATOR
          MOV     SI, SIZE TABLE    ;POINT SUBSCRIPT
                                    ;TO END OF TABLE
ADD_NEXT: SUB     SI, TYPE TABLE    ;BACK UP ONE ELEMENT
          ADD     AX, TABLE[SI]     ;ADD ELEMENT
          LOOP    ADD_NEXT          ;UNTIL CX=0
          ;AX CONTAINS SUM
```

図 6・13 データの属性と，ADD 命令のループ動作の例

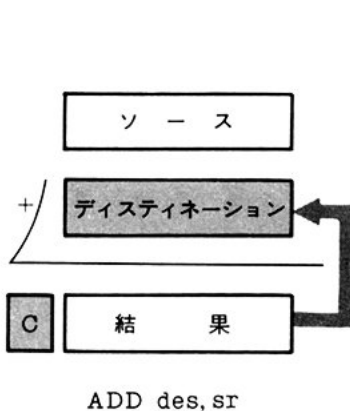
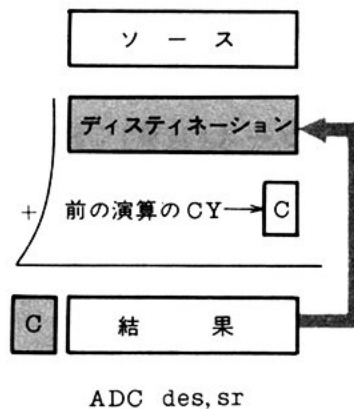
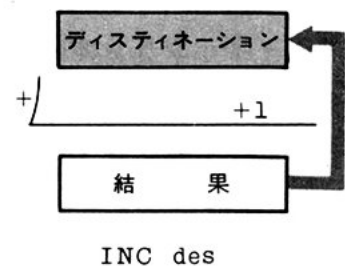



図 6・14 加算命令（1） 図 6・15 加算命令（2） 図 6・16 加算命令（3）

Mnemonics © Intel, 1978

〔2〕 **加算命令**

ADD ディスティネーション，ソース：ADDは，二つのオペランドの和が，ディスティネーションオペランドと置換される．これらのオペランドとしては符号付きまたは符号なしの2進数が可能で，AF, CF, OF, PF, SFおよびZFに影響を与える．

ADC ディスティネーション，ソース（キャリー付き加算）：ADCは，二つのオペランドの和を求め，もしCFフラグがセットされている場合は，それにさらに1を加算し，その結果はディスティネーションオペランドと置換される．オペランドとしては，ともに符号付きまたは符号なしが可能で，AF, CF, OF, PF, SFおよびZFに影響を与える．これは前の演算結果のキャリーを含めた演算ができるので，16ビット以上の演算に使用できる．

INC ディスティネーション：INCは，ディスティネーションに1を加算する．オペランドとしてはバイトまたはワードが可能で，AF, OF, PF, SF, およびZFに影響を与える．

AAA（加算のためのASCII調整）：AAAは，ALレジスタの内容をアンパックの10進数に変換し，その上位ニブルに0を入れる．AFおよびCFに影響を与え，OF, PF, SF, およびZFは不定になる．

DAA（加算のための10進調整）：DAAは，パックされた二つの数を正しい10進数に調整する．ALレジスタの内容を変換し，パックされた10進数とし，ALレジスタに結果が残る．AF, CF, PF, SFおよびZFに影響を与える．

〔3〕 **減算命令**

SUB ディスティネーション，ソース（減算）：SUBは，ソースオペランドがディスティネーションオペランドから減じられ，結果がディスティネーションオペランドに入る．オペランドはバイト/ワードが可能で，AF, CF, OF, PF, SFおよびZFに影響を与える．

SBB ディスティネーション，ソース（ボロー付き減算）：SBBは，前の演算で生じたボローを考慮した減算で，もしCFがセットされているとさらに1減じる．この命令は16ビット以上の減算に使用される．

DEC ディスティネーション（1減少）：DECは，ディスティネーションから

* 一つのアンパックのBCD加算の結果としてのALの値を補正する.

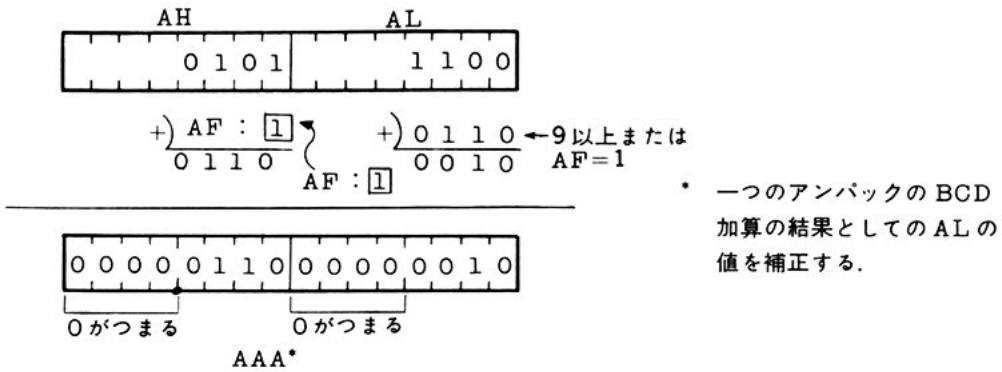



図 6・17 加算のためのASCII補正

* パックのBCD加算の結果としてのALの値を補正する.

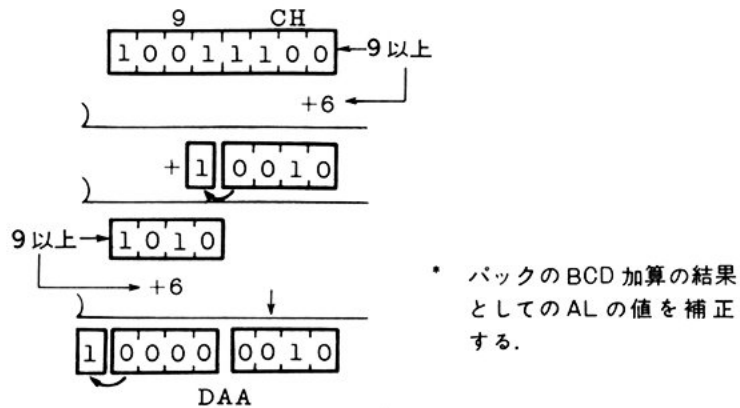



図 6・18 加算のための10進補正

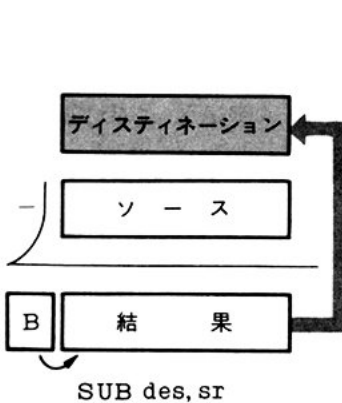
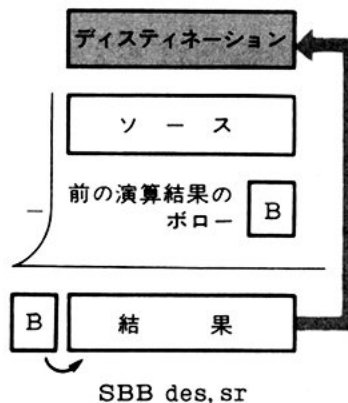
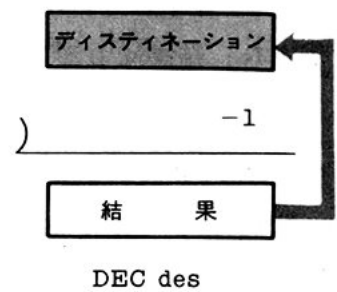



図 6・19 減算命令(1)　図 6・20 減算命令(2)　図 6・21 減算命令(3)

1減ずる．AF, OF, PF, SF および ZF に影響を与える．

NEG ディスティネーション（符号反転）：NEGは，0からディスティネーションオペランドを減じ，結果をディスティネーションに入れる．これはその数の2の補数の作成を行う．AF, CF, OF, PF, SF および ZF に影響を与える．

CMP ディスティネーション，ソース（比較）：CMPは，ディスティネーションからソースを減じ，その結果は返さず（不変）にフラグだけを更新する．この命令に続く条件ジャンプによりその結果を判定できる．AF, CF, OF, PF, SF および ZF に影響を与える．

AAS（ASCII Adjust for Subtraction）：AASは，前の減算結果としての二つの10進アンパックのオペランドをASCIIコードに調整する．この場合のディスティネーションはALレジスタとして指定されていなければならず，結果もアンパックの10進数としてALに残る．AFおよびCFのみ更新する．OF, PF, SF および ZF は不定になる．

DAS（Decimal Adjust for Subtraction）：DASは，前の減算結果としての二つのパックされた10進オペランドの結果を補正する．ディスティネーションとしてはALを指定しなければならず，結果もパックされた10進ディジットのペアとしてALに残る．AF, CF, PF, SF および ZF に影響を与える．OF は不定になる．

［4］ 乗除算命令

MUL ソース（乗算）：MULは，ソースオペランドとアキュムレータの符号なし乗算を実行する．ソースがバイトの場合はレジスタALとの間の乗算となり，結果はAHおよびALに返される．ソースがワード値の場合はレジスタAXとの間で乗算され，結果はレジスタDXおよびAXに返される．CFおよびOFは結果の上位半分（バイト演算ではAH，ワード演算ではDX）が0でない場合にセットされ，AF, PF, SF および ZF は不定になる．

IMUL ソース（整数乗算）：IMULは，ソースオペランドとアキュムレータの符号付き乗算を実行する．その他の動作はMULと同じである．結果の上位半分が，その結果の下位半分の符号拡張になっていない場合にCFおよびOFがセットされる．すなわち，CFおよびOFがセットされている場合は，AHまたはDXが結果

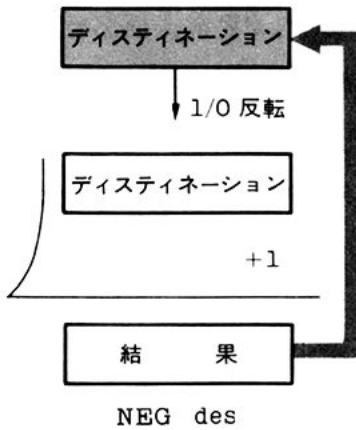



図 6・22 符号反転命令

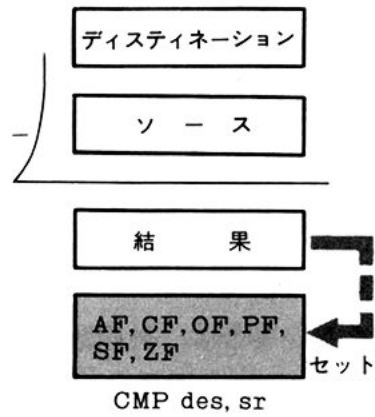



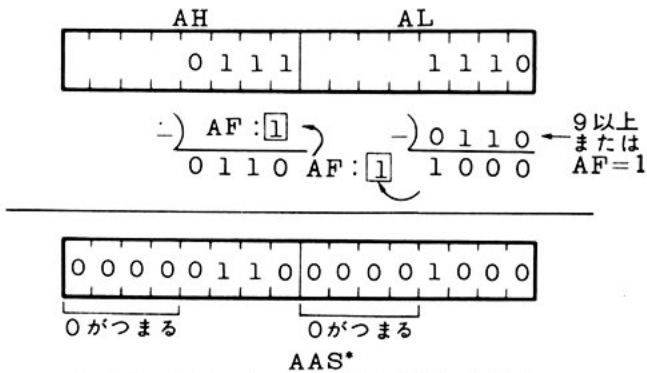
図 6・23 比較命令

* 二つのアンパックの BCD 演算の結果としての AL を補正する.

図 6・24 減算のための ASCII 補正

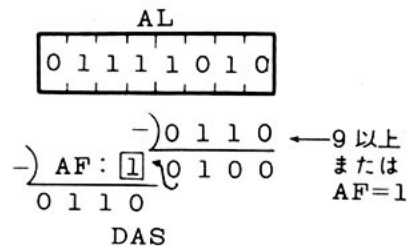



図 6・25 減算のための 10 進補正

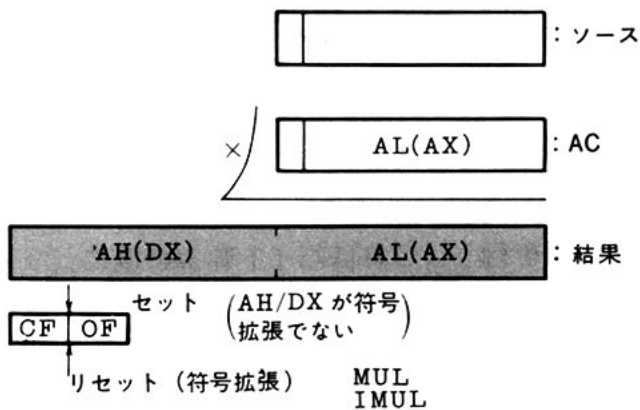



図 6・26 乗算命令

Mnemonics © Intel, 1978

の上位ディジットを含んでいることを示す．AF, PF, SFおよびZFは不定になる．

AAM（ASCII Adjust for Multiply）：AAMは，前の乗算結果としての二つのアンパックの10進オペランドを補正する．調整される数値はAHおよびALから持ってこられ，結果もAHおよびALに返される．PF, SFおよびZFに影響を与え，AF, CFおよびOFは不定になる．

DIV ソース（除算）：DIVは，アキュムレータの値をソースで割算する．ソースがバイトの場合は，被除数はAHおよびALにあるものと仮定される．結果としての商はALに，余りはAHに返される．ソースがワード値の場合は，被除数はAXおよびDXと仮定され，商はAXに，余りはDXに返される．結果がディスティネーションレジスタの範囲（バイトソースの場合FFH，ワードソースの場合FFFFH）を超えるとタイプ0の割込みがかかり，結果は不定となる．AF, CF, OF, PF, SFおよびZFは不定になる．

IDIV ソース（整数除算）：IDIVは，アキュムレータの内容をソースオペランドで符号付きの割算をする．動作はDIVに同じであるが，バイト除算の場合の商の正の最大値は＋127（7FH），負の最小値は－127（81H）となる．また，ワード除算では正が＋32767（7FFFH），負は－32767（8001H）である．商がこの範囲を超えるとタイプ0の割込みがかかり，結果は不定となる．AF, CF, PF, SFおよびZFは不定になる．

AAD（ASCII Adjust for Division）：AADは，二つのアンパックの10進オペランドを割算する前に，その商が有効なアンパックの10進数になるようにAL中の値の修正を行う．除算に先立ちAHを0にしておかなければならず，商はALに，余りはAHに返され，ともに上位ニブルは0となる．PF, SFおよびZFに影響を与え，AF, CFおよびOFは不定になる．

CBW（バイトからワードへの変換）：CBWは，ALレジスタ中のバイトをAHレジスタを通じての符号拡張を行う．これはバイト除算の前に，ワードの被除数を作成するのに使われる．フラグへの影響はない．

CWD（ワードからダブルワードへの変換）：CWDは，AXレジスタ中のワード値を，DXレジスタを通して符号拡張を行う．これはワード除算の前に，ダブルワードの被除数を作成するのに使われる．フラグへの影響はない．

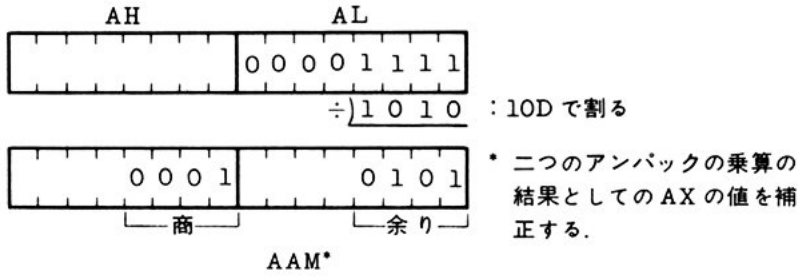



図 6・27 乗算のための ASCII 補正

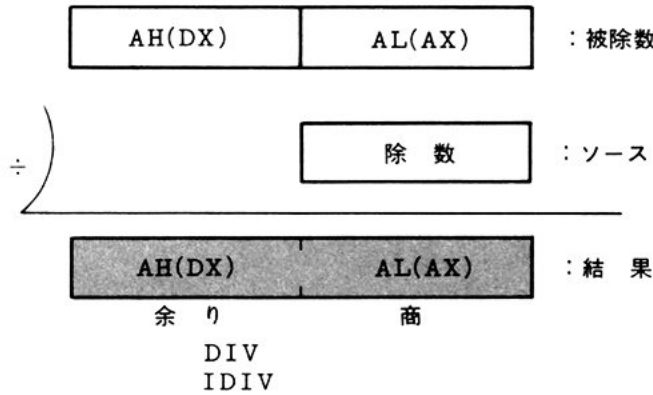



図 6・28 除算命令

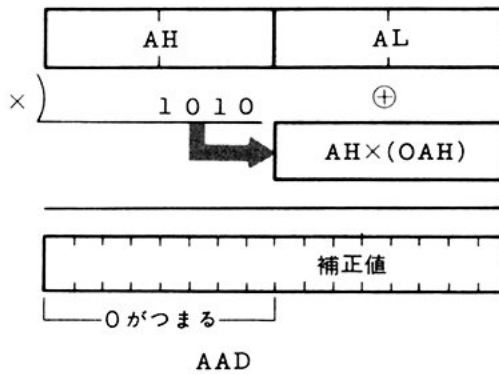



図 6・29 除算のための ASCII 補正

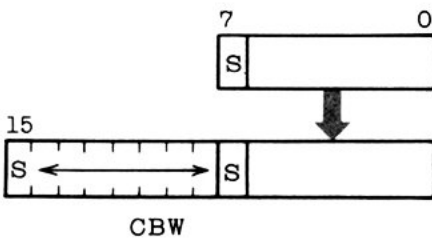
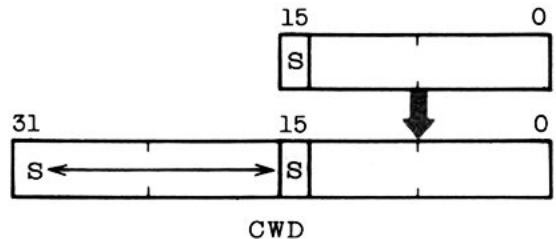



図 6・30 バイトからワードへの変換　　図 6・31 ワードからダブルワードへの変換

6・4 ビット操作命令

ビット操作命令には論理演算，シフト，およびローテートがある．

NOT ディスティネーション：NOTは，バイトまたはワードオペランドをビットごとに反転させる．

AND ディスティネーション，ソース：ANDは，二つのオペランドの論理積を実行し，結果をディスティネーションオペランドに返す．対応するビットがともに1の場合に1，一方が0の場合は0になる．

OR ディスティネーション，ソース：ORは，二つのオペランドの論理和を実行し，結果をディスティネーションオペランドに返す．対応するビットのいずれか，あるいは両方が1のときに1になる．

XOR ディスティネーション，ソース：XORは，二つのオペランドの排他的論理和を実行し，結果をディスティネーションオペランドに返す．対応するビットが相反する場合に1，同じ場合に0になる．

TEST ディスティネーション，ソース：TESTは，二つのオペランドの論理積を実行するが，結果は返さずにフラグだけを返す．二つのオペランドに対応するビットが存在する場合にフラグがセットされ，その後に続くJNZ命令によりその結果の判定を行う．

SHL/SAL ディスティネーション，カウント（左論理シフト/左算術シフト）：SHL/SALは，ディスティネーションで指定されたバイト/ワードをカウントオペランドで指定されたビット数だけ左にシフトさせ，右側からは0が入り込む．この二つの命令は同じ動作を実行する．

SHR ディスティネーション，カウント（論理右シフト）：SHRは，ディスティネーションオペランド中のビットをカウントオペランド中で指定されたビット数だけ右にシフトし，左側からは0がつまる．

SAR ディスティネーション，カウント（算術右シフト）：SARは，ディスティネーションオペランド中のビットをカウントオペランド中で指定されたビット数だけ右にシフトし，最上位ビット（符号ビット）は左にシフトして保存される．

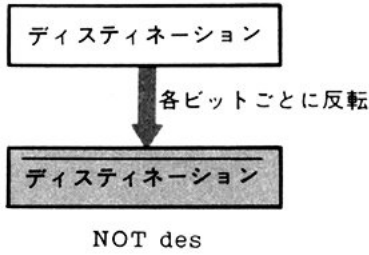



図 6・32　ビット反転

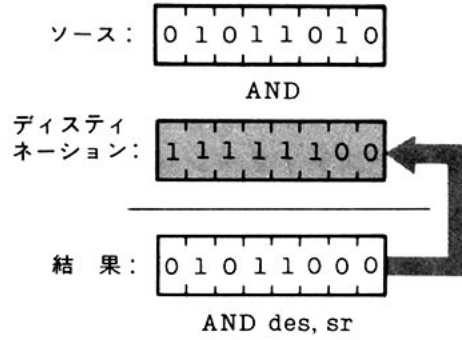



図 6・33　論理積命令

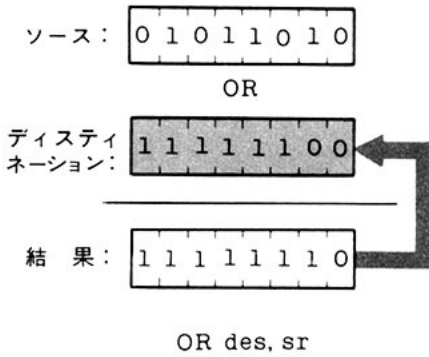



図 6・34　論理和命令

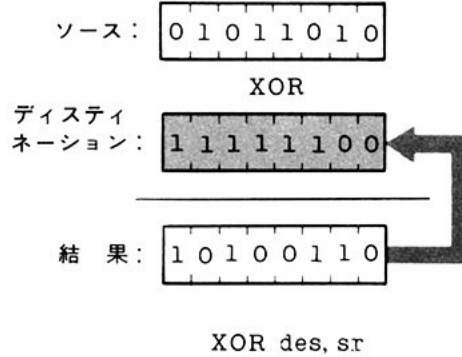



図 6・35　排他的論理和

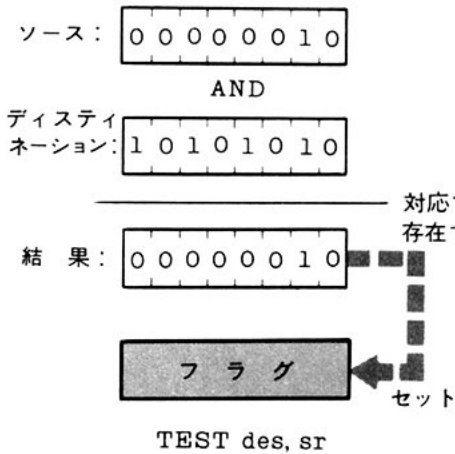



図 6・36　テスト命令

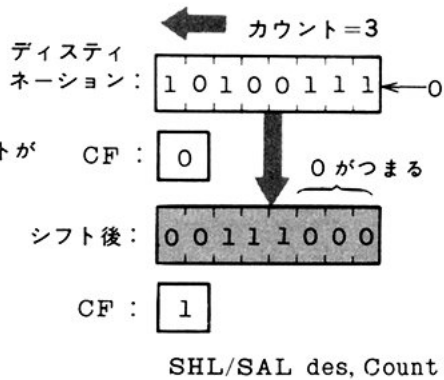



図 6・37　左シフト命令

ROL ディスティネーション，カウント（左回転）：ROLは，ディスティネーションのバイトまたはワードを，カウントオペランドで指定されたビット数だけ左に回転する．

ROR ディスティネーション，カウント（右回転）：RORは，右回転を行う以外，動作はROLと全く同じである．

RCL ディスティネーション，カウント（キャリーを通しての左回転）：RCLは，キャリー（CF）が左回転のループの中に入り，ディスティネーションの上位からはみ出したビットは，下位ビットから入り込む．それ以外の動作はROLに同じである．

RCR ディスティネーション，カウント（キャリーを通しての右回転）：RCRは，ビットが右回転になる以外はRCLと同じである．

カウント＝3

ディスティネーション：10100111

CF：0

0がつまる

00010100

CF：1

SHR des, Count

図 6・38 右シフト命令（1）

カウント＝3

0 10100111

CF

1 11110100

CF

SAR des, Count

図 6・39 右シフト命令（2）

カウント＝3

CF

0 11010111 ：ディスティネーション

0 10111110 ：結 果

ROL des, Count

図 6・40 左回転命令（1）

カウント＝3

0 11010111 ：ディスティネーション

1 11111010 ：結 果

ROR des, Count

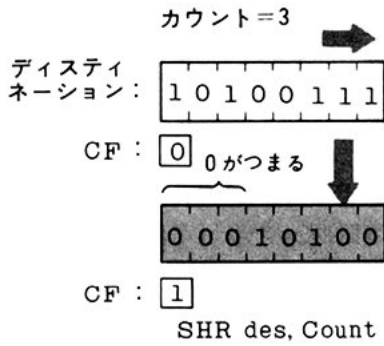
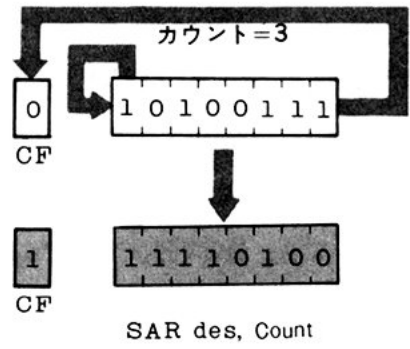
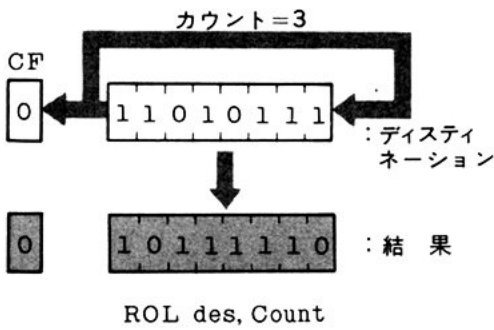
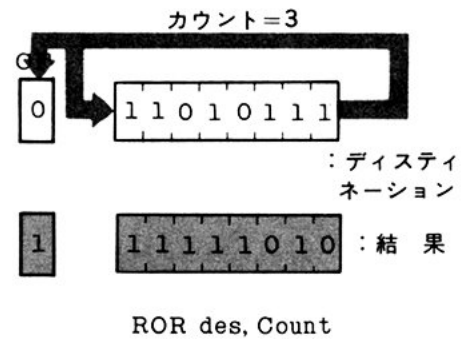



図 6・41 右回転命令（1）

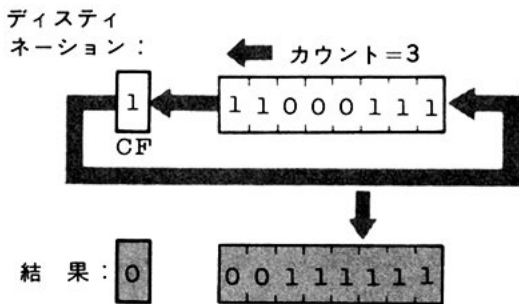
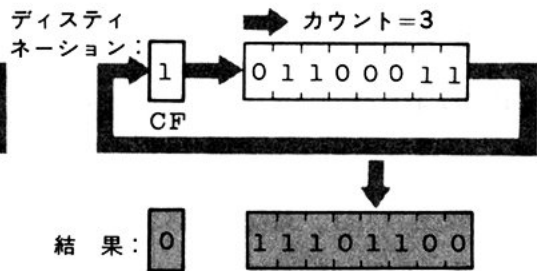



図 6・42 左回転（2）

図 6・43 右回転（2）

6・5 ストリング命令

ストリング命令は命令の前に付加する1バイトのプリミティブと呼ばれる特殊な命令により，MOVE, COMPARE, SCAN 等の動作を連続して実行させることができるもので，データのブロックムーブや特定のコードのサーチなどに有効である．

このストリング動作にはソースストリング側のセグメントレジスタとしてはデータセグメント（DS）が使用され，ディスティネーション側はエックストラセグメント（ES）にデフォルトとして決まっている（表2・1参照）．そして，ストリング動作中にアドレスを順番に更新してゆくレジスタとしては，ソース側がSI，ディスティネーション側はDIレジスタとなっている．これらの使用法に関しては図6・44参照のこと．

ストリング命令ではDF（ディレクションフラグ）が0または1のどちらに設定されているかにより，SIおよびDIの値を自動的に増/減させ，バイトストリングの場合は1ずつ，ワードの場合は2ずつ調整される．また，ストリング命令の繰返しの数をカウントするためにはCXレジスタが使用され，1回の実行ごとに1ずつ減じられる．したがって，ストリング動作に先立ち，CXに繰返し回数を設定する必要がある．

REP/REPE/REPZ/REPNE/REPNZ：以上5種類の命令は，その後に続くストリング命令の繰返しをコントロールするプリフィックスバイトである．

REP（繰返し）は，MOVSおよびSTOS（後述）と結合して使用され，CXが0でない間，その動作を繰り返す．

REPE（等しい間繰返し）およびREPZ（0の間繰返し）は，類似した働きをし，CMPSおよびSCAS（後述）とともに使用される．REPNE（等しくない間繰返し）およびREPNZ（0でない間繰返し）も動作は前者とほぼ同じであるが，繰返し動作に先立ち，前者はZFをセットしておかなければならないのに対し，後者はそれをクリヤしておかなければならない点が異なる．

MOVS ディスティネーションストリング，ソースストリング：MOVSは，ソ

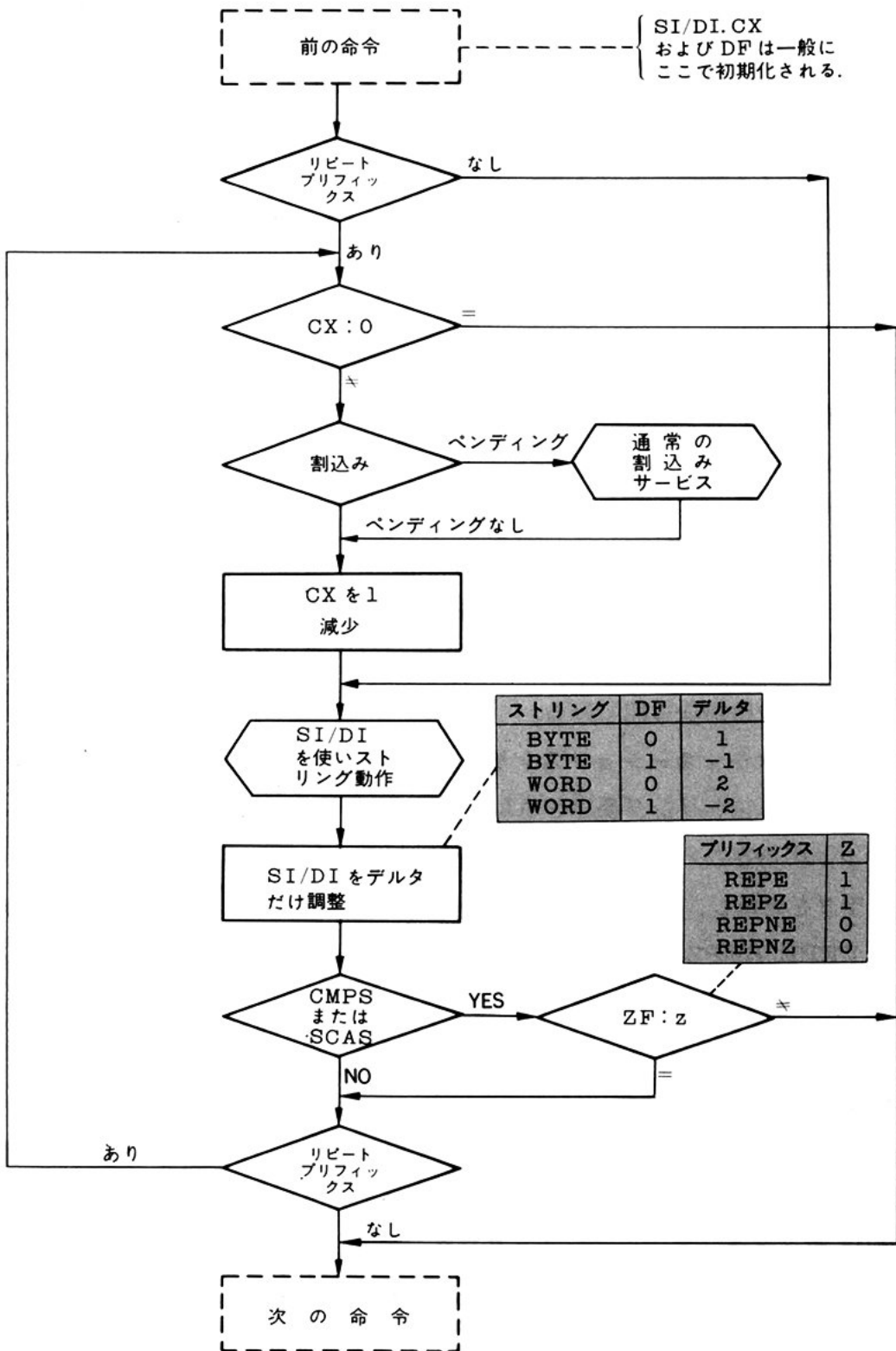



図 6・44 ストリング動作のフローチャート

ースストリング（SIで指定）からのバイトまたはワードをディスティネーションストリング（DIで指定）に転送し，SIおよびDIが次のストリング要素を指示するよう更新する．前述のREPプリフィックスとともに使用され，メモリ・メモリ間のブロック転送を実行する．

MOVSB/MOVSW：この命令はMOVSの代替となるもので，MOVの対象となるデータがバイトあるいはワードであることを明確に規定するものである．動作は同じ．

CMPS ディスティネーションストリング，ソースストリング：CMPS（ストリングの比較）は，ソースストリングのバイト/ワードからディスティネーションのバイト/ワードを引算し，その結果に応じてフラグをセットし，SIおよびDIを次のストリング要素を指示するように更新する．この命令を実行するとAF,CF,OF,PF,SFおよびZFが影響され，その後に続く条件ジャンプの命令で大小の判定を行うことができる．また，前述のREPE/REPZおよびREPNE/REPNZと組み合わせて，連続したストリングの比較が可能で，各ストリング要素の一致または不一致の検出が可能である．

SCAS ディスティネーションストリング（ストリング走査）：SCASは，DIでアドレスされるストリング要素をAL（バイトストリング）またはAX（ワードストリング）の内容から引算し，その結果に応じてAF,CF,OF,PF,SF およびZFフラグを更新し，DIを次のストリング要素を指示するよう更新する．この命令は，REPE/REPZおよびREPNE/REPNZプリミティブと組み合わせて，アキュムレータの内容との一致/不一致のストリング要素を見つけるのに使用される．

LODS ソースストリング（ロードストリング）：LODSは，SIでアドレスされたバイトまたはワードをALまたはAXに転送し，SIをストリング中の次の要素を指すよう更新する．リピート動作はない．

STOS ディスティネーションストリング（ストアストリング）：STOSは，DIによりアドレスされるストリング要素にバイトデータALまたはワードデータAXを転送し，DIを次のストリングロケーションのために更新する．この命令のリピート動作はストリングをある定数で満たすのに有効である．

図 6・45 ストリング命令一覧

表 6・3 ストリング動作におけるレジスタ/フラグの使用

レジスタ/フラグ	用 途
SI	ソースストリングのためのインデックス
DI	ディスティネーションストリングのためのインデックス
CX	繰返しカウンタ
AL/AX	◦走査する値 ◦LODS のためのソース ◦STOS のためのディスティネーション
DF	0：SI, DI の自動的インクリメント 1：SI, DI の自動的デクリメント
ZF	走査/比較のターミネータ

6・6 プログラム転送命令

プログラム転送命令は，CSおよびIPの内容を変えることによりプログラムの流れを変えるものであり，無条件転送,条件付き転送,繰返しコントロール，および割込みコントロール命令等がある．

CALL プロセデュア名：CALLは，返り先の情報をスタックにセーブして，サブルーチンに分岐する．CALLには次の4通りの場合がある．

（1） **セグメント内直接 CALL**：SPを2減じて，IPをスタックにプッシュした後，CALL命令のオペランドからのターゲットのプロセデュアに対する相対変位が，命令ポインタに加えられ，自己相対のサブルーチンコールを実行する．

（2） **セグメント内間接 CALL**：SPを2減じ，IPをスタックにプッシュした後，ターゲットとなるプロセデュアのオフセットは，その命令で参照されるメモリまたはレジスタから得られ，IPを置き換えてコールする．

（3） **セグメント外直接 CALL**：SPを2減じ，CSをスタックにセーブした後，そのCSは命令中に含まれているセグメントで置き換えられる．次に，SPは再び2減じ，IPをスタックにセーブし，それは命令オペランド中のオフセットで置き換えられる．

（4） **セグメント外間接 CALL**：SPを2減じ，CSをセーブした後，CSはその命令により参照されるダブルワードのメモリポインタの2番目のワードにより置き換えられる．次に，SPは再び2減じ，IPをセーブした後，それは命令により参照されたダブルワードのポインタの最初のワードにより置き換えられ，コールを実行する．

［1］ 無条件転送命令

RET オプションの POP値：RETは，CALLで呼び出されたプロセデュアからの復帰で，CALLがセグメント内（NEAR）か，セグメント外（FAR）かにより2通りがある．RETは，TOS（スタックの先頭）にあるワードをIPにポップし，SPを2増加する．もしセグメント外RETの場合は，スタックの新しい先頭にあるワードがCSレジスタ中にポップされ，SPは再び2増加される．また

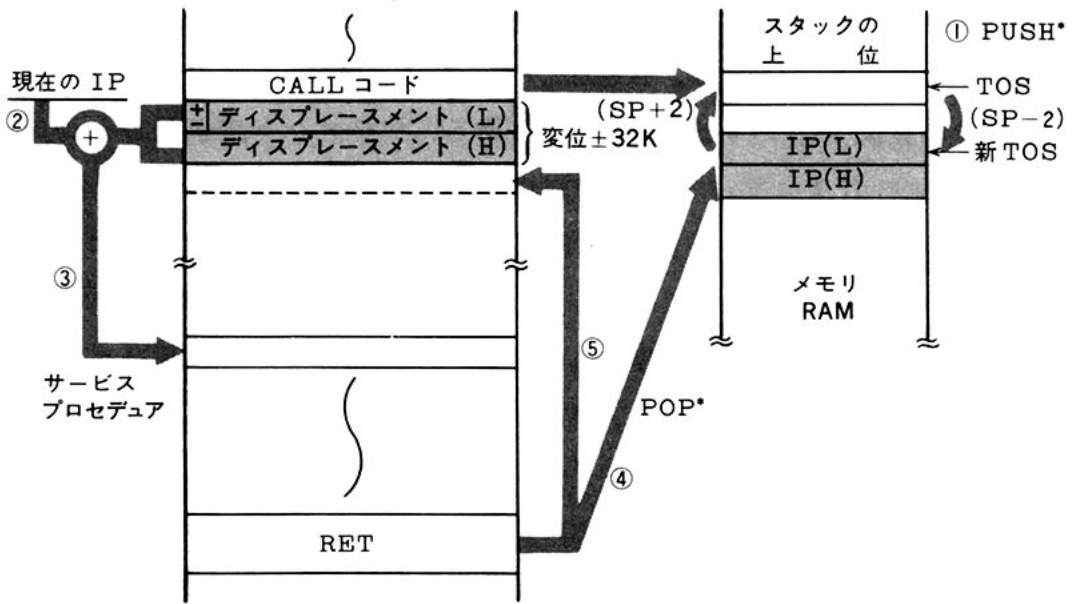



図 6・46 セグメント内直接コール/ジャンプ (1)

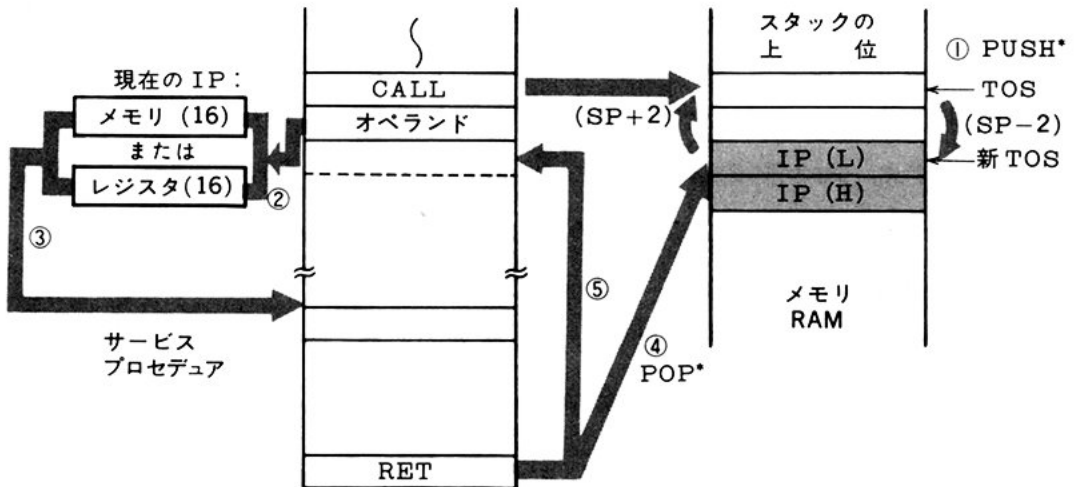



図 6・47 セグメント内間接コール/ジャンプ (2)

オプションのPOP値が指定されている場合は，その値をSPに加える．

JMP ターゲット：JMPは，無条件ジャンプで，CALLと同様にセグメント内/外および直接/間接の4種類の組合せがある．セグメント内直接ジャンプはJMP命令からのターゲットの相対変位を加えることによりIPを変化させ，ジャンプする．ターゲットがそのJMP命令から+127および−128バイト以内の場合は，アセンブラは自動的に判断して，SHORT JMPと呼ばれる2バイトのコードを発生する．それ以外の場合はNEAR JMPとなり，±32Kバイトのジャンプが可能である．このジャンプは自己相対で，コードの位置に影響されないリロケータブルなプログラミングができる．

セグメント内間接ジャンプは，メモリまたはレジスタを通して間接的に行われる．これらの場合は，メモリまたはレジスタの内容がIPと置き換わる．セグメント外直接ジャンプは，命令のオペランドに含まれる値がIPおよびCSと置換される．セグメント外間接ジャンプはメモリを通して行われ，その命令により参照されるダブルワードのポインタの最初のワードがIPを，2番目のワードがCSを置換することによりジャンプを実行する．

〔2〕 **条件付き転送命令**　条件付き転送は，この命令に先立ち実行されたプログラムにより変化したCPUのフラグを調べ，その条件が“真”の場合は，その命令で指定されたアドレスに分岐し，“偽”の場合はその命令の次にコントロールが移される．すべての条件ジャンプはSHORTで，その命令から+127および−128バイト以内である．これらの命令の一覧表を表6·4に示す．

〔3〕 **繰返しコントロール**　繰返しコントロールは，ソフトウェアの繰返しループを可能にするもので，CXレジスタをそのカウンタとして使用する．この繰返し命令は自己相対で，その命令から+127および−128以内でなければならない．

LOOP ショートラベル：LOOPは，CXを1減じて，CXが0でない場合は，コントロールを指定のアドレスに移し，さもない場合はLOOPの後の命令を実行する．

LOOPE/LOOPZ ショートラベル（等しいかあるいは0の間ループ）：LOOPE/LOOPZは，CXを1減じて，CXが0でなく，そしてゼロフラグがセットされた場

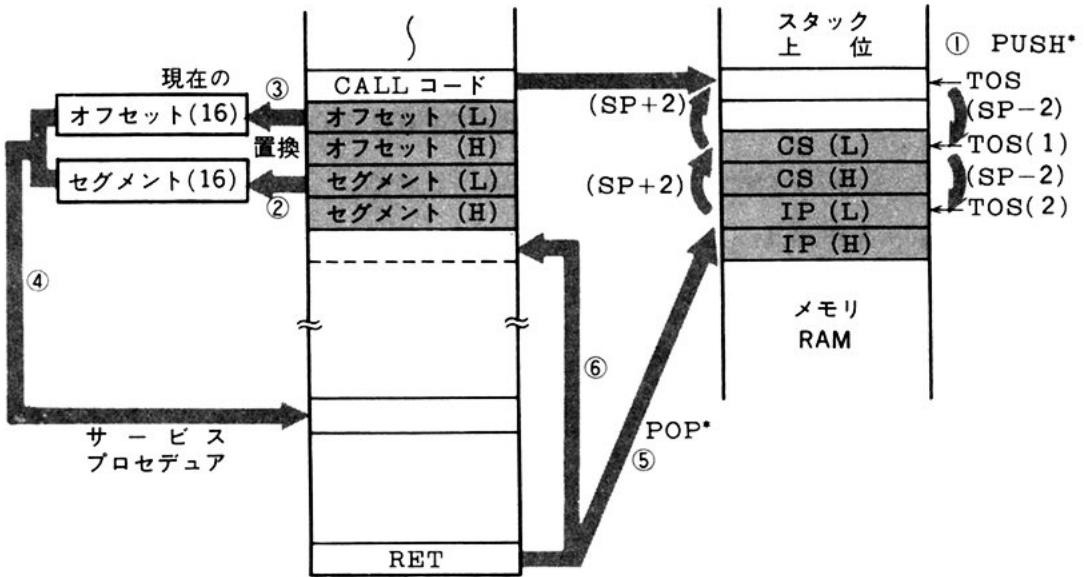



図 6・48 セグメント外直接コール/ジャンプ（3）

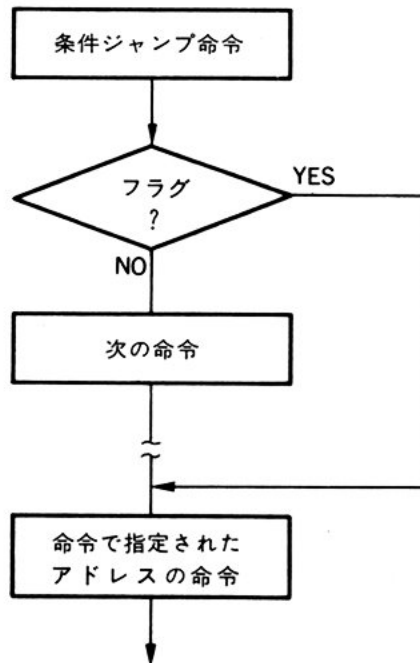



図 6・49 条件付きジャンプのフロー

合にコントロールを指定のアドレスに移し，そうでない場合は，その命令の次の命令を実行する．

LOOPNE/LOOPNZ ショートラベル：LOOPNE/LOOPNZ は，条件が逆になった以外は，基本的な動作は前の命令と同じである．

JCXZ ショートラベル（CX が 0 の場合にジャンプ）：JCXZ は，CX が 0 の場合に指定のアドレスにジャンプする．

〔**4**〕 **割込み命令**　割込み命令は，ハード的な割込みのほかにプログラムにより割込み処理ルーチンの起動が可能である．この場合の動作はハードウェアによるものとほぼ同じであるが，ソフトウェアによる場合は割込みアクノレージバスサイクルが実行されない点が異なっている．

INT 割込みタイプ：INT は，オペランドの割込みタイプにより指定される割込みプロセデュアを活性化する．INT は，スタックポインタを 2 減じ，フラグをスタックにプッシュし，そしてシングルステップおよびマスカブル割込みを禁止するためにトラップおよび割込みイネーブルフラグをクリヤする．次に，SP を再び 2 減じ，CS レジスタをスタックにプッシュする．割込みポインタのアドレスは 8259A から受け取った割込みタイプに 4 を掛けることによって計算され，割込みポインタの 2 番目のワードは CS を置換する．SP をさらに 2 減じ，IP をスタックにプッシュした後，IP は割込みポインタの最初のワードで置き換えられる．このソフトウェア割込みは，オペレーティングシステムからのサービス要求としての“スーパーバイザコール”として使用することができる．

INTO（オーバフロー割込み）：INTO は，演算結果としてオーバフローフラグがセットされた場合にソフトウェアの割込みを発生し，そうでない場合は割込みを発生せずにそのまま次の命令へ進む．INTO は，割込みポインタテーブルの 10H を通じて割込みプロセデュアにコントロールを移す．

IRET（割込みからの復帰）：IRET は，IP, CS およびフラグをスタックからポップした後，割込みの発生した点の次の命令にコントロールを移す．割込みプロセデュアからの復帰にはすべて，IRET を使用しなければならない．

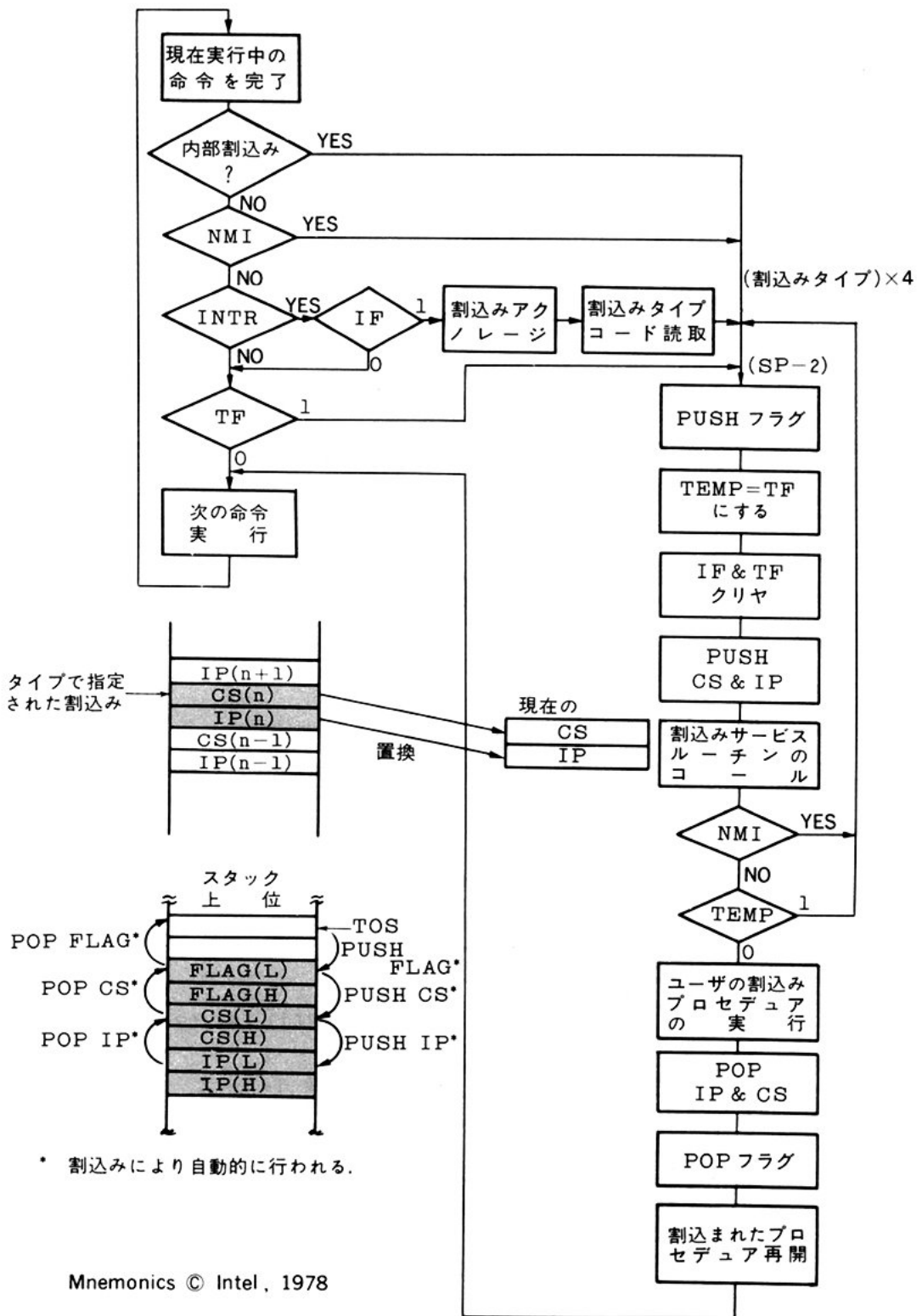



図 6・50　割込み処理シーケンス

表 6・4　条件付きジャンプ一覧表

命令ニーモニック	テスト条件	意味
JA/JNBE	(CF または ZF)=0	以上/以下でなく，等しくもない
JAE/JNB	CF=0	以上または等しい/以下でない
JB/JNAE	CF=1	以下/以上でなく，等しくもない
JBE/JNA	(CF または ZF)=1	以下または等しい/以上でない
JC	CF=1	キャリー
JE/JZ	ZF=1	等しい/ゼロ
JG/JNLE	((SF XOR OF) または ZF)=0	より大きい/より小さくなく，等しくもない
JGE/JNL	(SF XOR OF)=0	より大きいか，等しい/より小さくない
JL/JNGE	(SF XOR OF)=1	より小さい/より大きくはなく，等しくもない
JLE/JNG	((SF XOR OF)OR ZF)=1	より小さいか等しい/より大きくない
JNC	CF=0	キャリーなし
JNE/JNZ	ZF=0	等しくない/ゼロでない
JNO	OF=0	オーバフローなし
JNP/JPO	PF=0	パリティなし/奇数パリティ
JNS	SF=0	符号なし（正）
JO	OF=1	オーバフロー
JP/JPE	PF=1	パリティ/偶数パリティ
JS	SF=1	符号あり（負）

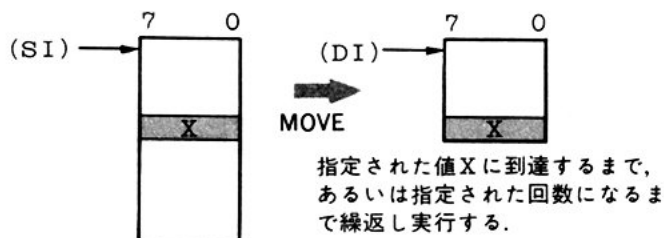



図 6・51　ストリング動作によるブロックムーブの例

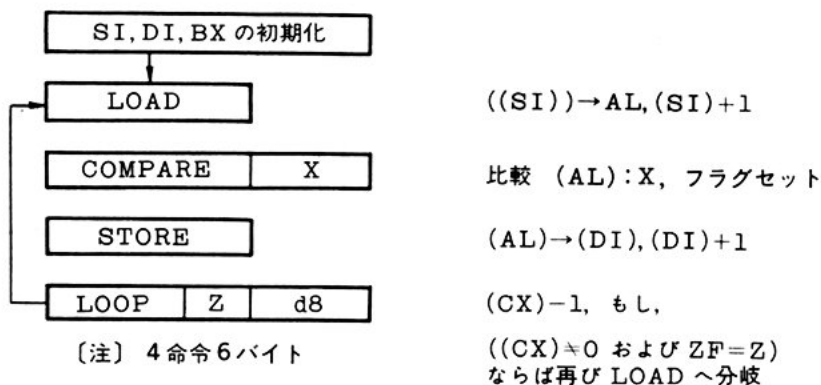



図 6・52　ストリングプリミティブによる LOOP 動作例

6・7 プロセッサコントロール命令

この一連の命令は，CPUフラグのコントロール，およびCPUを外部事象に同期させるのに使用される．

CLC（クリヤキャリーフラグ）：CLCは，キャリーフラグ（CF）をクリヤする．他のフラグへの影響はない．

CMC（コンプリメントキャリーフラグ）：CMCは，キャリーフラグの状態を反転させる．他のフラグへの影響はない．

STC（セットキャリーフラグ）：STCは，キャリーフラグを1にセットする．他のフラグへの影響はない．

CLD（クリヤディレクションフラグ）：CLDは，ディレクションフラグ（DF）を0にセットし，ストリング命令の場合に，インデックスレジスタSIおよびDIを自動的に増加する状態にする．

STD（セットディレクションフラグ）：STDは，ディレクションフラグを1にセットし，ストリング動作の場合に，インデックスレジスタSIおよびDIを自動的に減少させる状態にする．

CLI（クリヤ割込みイネーブルフラグ）：割込みイネーブルフラグ（IF）を0にし，INTR（マスカブル割込み）からの割込みを禁止する．NMI（ノンマスカブル割込み）は禁止にならず，ソフトウェア割込みとして扱われる．

HLT（ホルト）：HLTは，CPUをホルト状態にする．リセットまたは外部割込みがかかってくるまで中断の状態になる．

WAIT：CPUの$\overline{\text{TEST}}$端子がアクティブ（0）でない間待ち状態．

ESC外部OPコード，ソース：マルチCPUシステムにおいて，外部プロセッサ（8087等）が命令コード，メモリオペランドを8086/8088から受け取る機能を提供．外部プロセッサはシステムバスを監視し，ESCがフェッチされたときこのOPコードをとらえ，8086 CPUがメモリからメモリオペランドを読み出したときそれを横取りする（12・1節）．

LOCK：1バイトのプリフィックスで，8086/8088（マキシマムモード）が次の

命令の実行中にバスロック信号を出す原因になる.

NOP(ノーオペレーション):CPUが何もせず次の命令に移行する.

メモリ参照命令では,その命令の実行時間に,実効アドレス(**EA**;実際のメモリを指定するアドレス値)算出時間を加えたものがその命令の実際の実行時間となる.以下にその一覧表を示す.

実効アドレス算出要素	記 号 表 示	クロック数
ディスプレースメントのみ	DISP	6
ベースまたはインデックスのみ	(BX, BP, SI, DI)	5
ベースまたはインデックス+ディスプレースメント	(BX, BP, SI, DI) +DISP	9
ベース+インデックス	BP+DI, BX+SI	7
	BP+SI, BX+DI	8
ベース+インデックス+ディスプレースメント	BP+DI+DISP BX+SI+DISP	11
	BP+SI+DISP BX+DI+DISP	12

7. アドレッシングモード

8086のアドレッシングのおのおのについて解説する．レジスタを通じての間接アドレッシング，および命令中のディスプレースメントとの組合せによる相対アドレッシングの種々の組合せが可能である．また，インデックスレジスタは特定な命令と組み合わされて，ストリング動作などの強力なデータ処理機能を提供する．

7・1 レジスタおよび直接オペランド

レジスタに対する指定は，命令の中に直接エンコードされた形で組み込まれているので，その動作にはバスサイクルを必要としない．そして，処理はすべてCPU内部だけで行われるので命令がコンパクトになり，実行時間も短い．レジスタはソースおよびディスティネーションオペランドになることができ，両方ともレジスタであることも可能である．

イメディエート（直接）オペランドというのは命令中に含まれている定数データで，その値としては8または16ビットが可能である．このイメディエートデータもレジスタの場合と同様に，命令キューの中から直接持ってくることができ，バスサイクルを必要としないため，高速の実行ができ，この場合は当然のことながらソースオペランドだけである．

表 7・1 汎用レジスタの暗黙の使用

レジスタ	動　作
AX	ワード乗算，ワード除算 ワード I/O
AL	バイト乗算，バイト除算 バイト I/O，翻訳，10進演算
AH	バイト乗算，バイト除算
BX	翻　訳
CX	ストリング動作，ループ動作
CL	可変シフトおよび回転
DX	ワード乗算，ワード除算 間接 I/O
SP	スタック動作
SI	ストリング動作
DI	ストリング動作

7・2 メモリアドレッシングモード

メモリオペランドの動作はバスを通じてCPUとの間で転送しなければならないので，そのメモリアドレス指定のため命令は長くなり実行時間も長くかかる．

実行ユニット(EU)は，メモリオペランドのリード/ライトが必要になると，BIUにアドレスのためのオフセット値（変位）を送り，それにセグメントレジスタの内容を加算して20ビットのアドレスを発生し，その指定されたメモリオペランドをアクセスするバスサイクルを実行する．

〔1〕 **実効アドレス**　EUがメモリオペランドをアドレスするために計算するオフセットは実効アドレス(EA)と呼ばれ，符号なしの16ビット数で，その命令が含まれるセグメントの始まりからの距離を表している．EUが実効アドレスを算出するのにはいくつかの方法があり，その指定は命令中の第2番目のバイト中にエンコードされて含まれている．

図7・1にEUがEAを算出する系統図を示す．これらの要素の組合せにより8086/8088の多彩なアドレッシングモードが可能である．この場合のディスプレースメント（変位）は命令中に含まれ，8または16ビットが可能で，プログラム中のオペランド名（変数またはラベル）の位置（アドレス）に由来する．

また，BXおよびBPがEAを算出する場合のベースレジスタとして指定でき，同様にSIおよびDIはインデックスレジスタとしての使用が可能である．このベースおよびインデックスレジスタの内容はプログラム実行中に変更できるので，それによって異なったメモリロケーションをアクセスすることができる．

〔2〕 **メモリアドレッシング**

（a） **直接アドレッシング**　直接アドレッシングは，図7・2に示すように，その実効アドレスは命令のディスプレースメント部分から直接持ってくる．この直接アドレスは簡単な変数のアクセス等におもに使用される．

（b） **レジスタ間接アドレッシング**　図7・3に示すように，メモリオペランドの実効アドレスは，ベースレジスタまたはインデックスレジスタのうちの一つから直接持ってくる．LEAおよび算述演算命令がこのレジスタ値の変更に使用

〈シングルインデックス〉 〈ダブルインデックス〉

BX OR BP OR SI OR DI

BX OR BP

SI OR DI

命令中にエンコードされ組み込まれている

命令のオペランドで指定

命令オペランド

変位

実効アドレス

EU

命令の種類によりいずれかが使用される

CS 0000 OR SS 0000 OR DS 0000 OR ES 0000

BIU

物理的アドレス

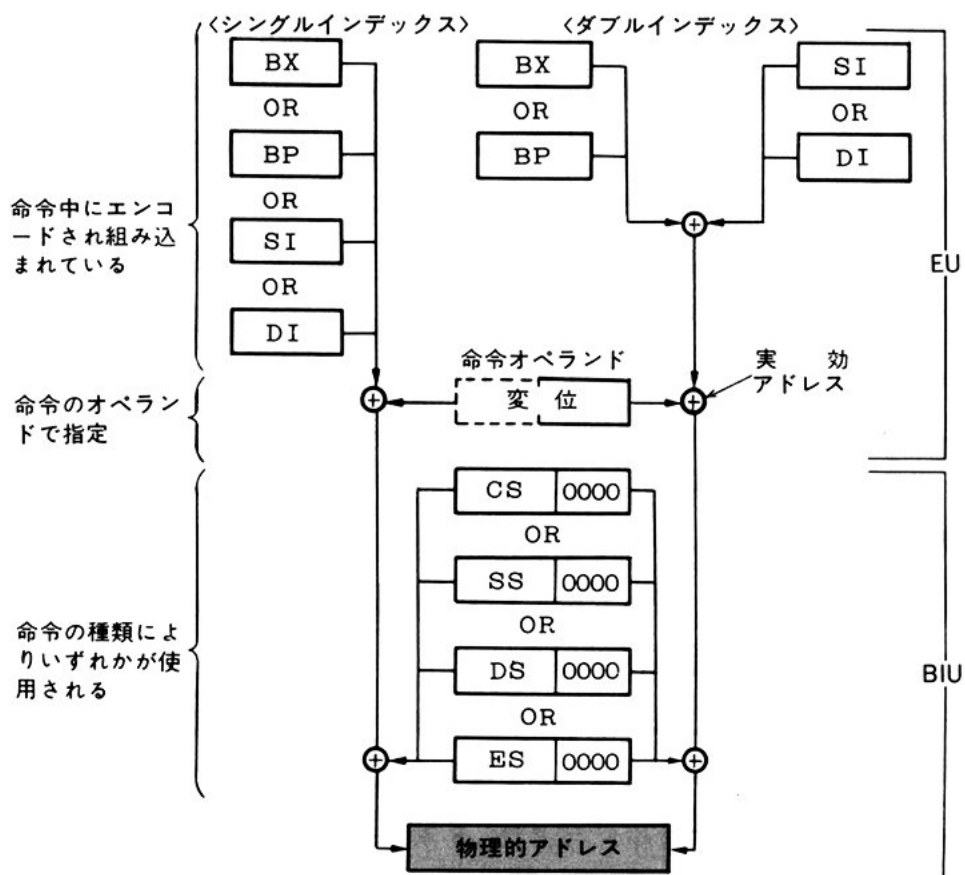



図 7・1 メモリアドレスの計算

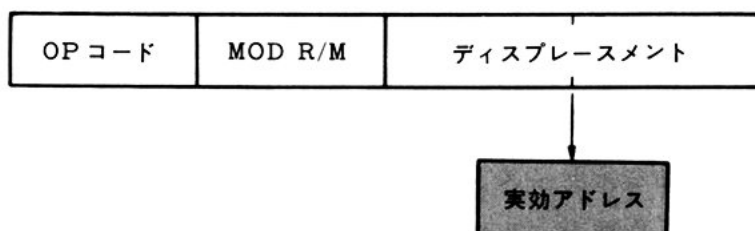



図 7・2 直接アドレッシング

される．すべての16ビット汎用レジスタがレジスタ間接アドレッシングに使用可能である．

（c）**ベースを持ったアドレッシング**　ベースを持ったアドレッシングの実効アドレスは図7·4に示すようにディスプレースメント値とBXまたはBPレジスタの内容との和になる．ベースレジスタとしてBPを指定すると，BIUはそのオペランドを現在のスタックセグメントから持ってくるので，スタックのデータをアクセスするための非常に便利な方法を提供する．また，このベースを持ったアドレッ

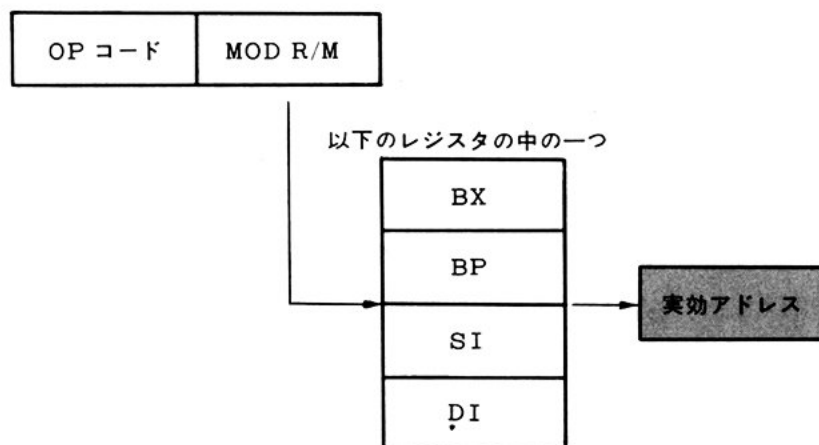



図7·3　レジスタ間接アドレッシング

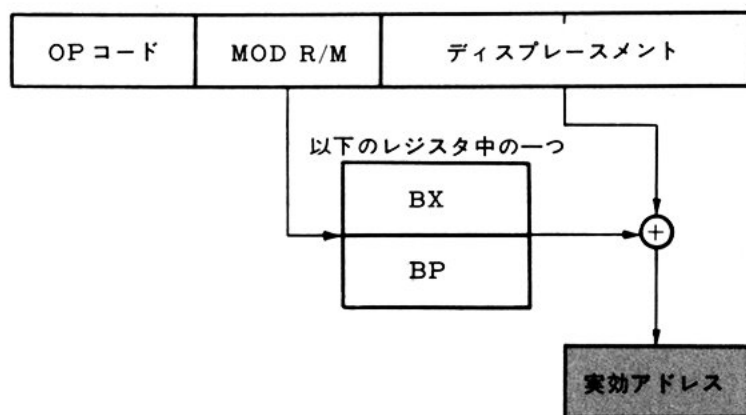



図7·4　ベースを持ったアドレッシング

シングを使ってメモリ中の異なった場所に位置するストラクチャをアドレスする例を図7·5に示す．ストラクチャのベースはベースレジスタで指示し，各要素はそのベースからのディスプレースメントで示される．したがって，同じストラクチャ中の別のデータはそのベースレジスタを変えてアクセス可能である．

（**d**）**インデックスアドレッシング**　この場合の実効アドレスは，ディスプレースメントとインデックスレジスタ（SIまたはDI）の和から計算される．このインデックスアドレッシングは，図7·7に示すように配列のアドレッシングに使用され，ディスプレースメントでその配列の始まりを示し，インデックスレジスタの値でそのおのおのの要素を指定する．

（**e**）**ベースを持ったインデックスアドレッシング**　これは，図7·8に示すようにベースレジスタ，インデックスレジスタおよびディスプレースメントの和から実効アドレスを算出するものである．ベースを持ったインデックスアドレッシングは，プロセデュアがスタック上に位置する配列をアドレスするための便利な方法を提供する．BPレジスタは，そのスタック上の基準点のアドレス（通常，そのプロセデュアがレジスタをセーブし，ローカルストレージを割り当てた後のスタックの先頭）を含み，その点からの配列の始まりのアドレスはディスプレースメントの値により表される．そして，インデックスレジスタは個々の配列要素をアクセスするのに使用される（図7·9）．また，ストラクチャやマトリクス中に含まれる配列も，このベースを持ったインデックスアドレッシングでアクセス可能である．

（**f**）**ストリングアドレッシング**　ストリング命令は，そのオペランドをアクセスするのに普通のメモリアドレッシングモードは使用せずに，図7·10のように暗黙のうちにインデックスレジスタを使用する．そしてこの場合，SIがソースストリングの最初のバイトまたはワードを，DIはそのディスティネーションストリングの最初を指示しているものと仮定している．連続したストリング動作の場合には，CPUが自動的にSIおよびDIの値を調整（増/減）する．

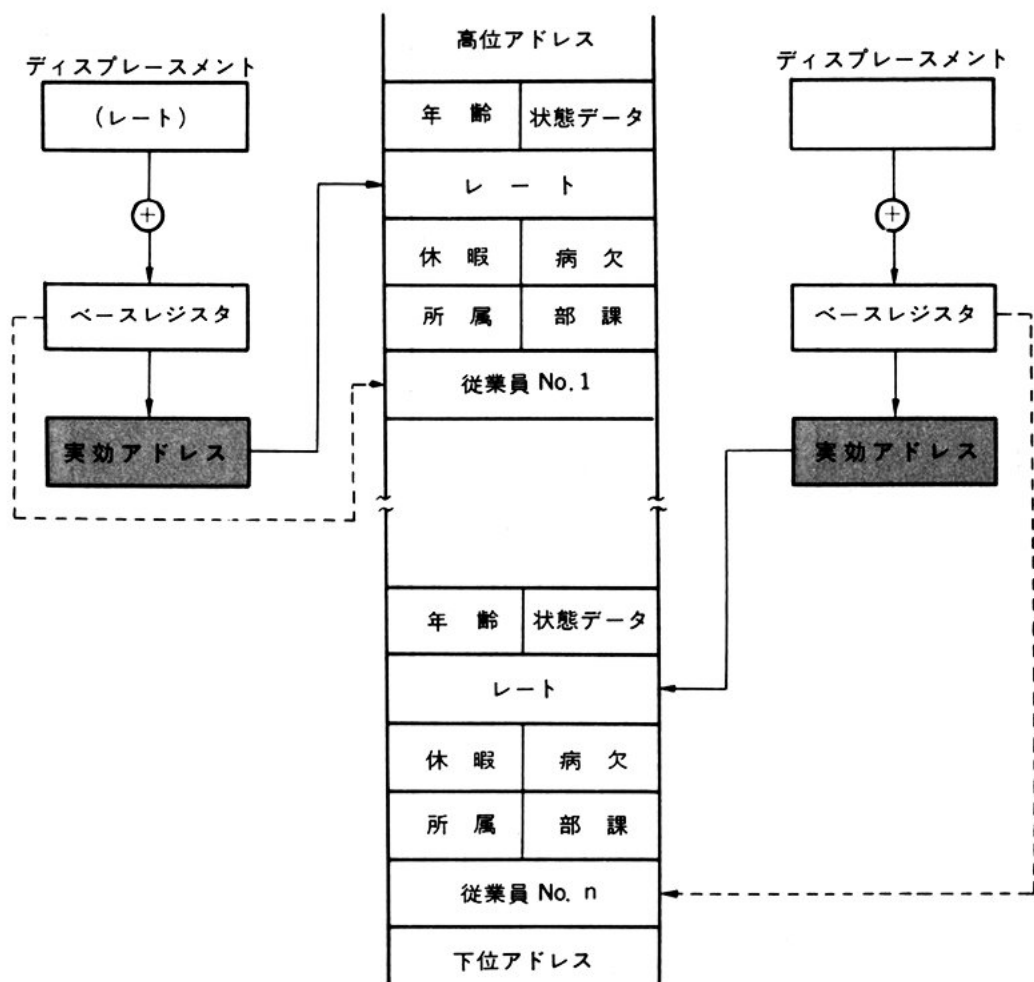



図 7・5 ベースを持ったアドレッシングによるストラクチャのアクセス

OP コード
MOD R/M
ディスプレースメント
以下のレジスタのいずれか
SI
DI
実効アドレス

図 7・6 インデックスアドレッシング

高位アドレス

ディスプレースメント

インデックスレジスタ

(14)

実効アドレス

配列 (8)
配列 (7)
配列 (6)
配列 (5)
配列 (4)
配列 (3)
配列 (2)
配列 (1)
配列 (0)

1ワード

ディスプレースメント

インデックスレジスタ

(2)

実効アドレス

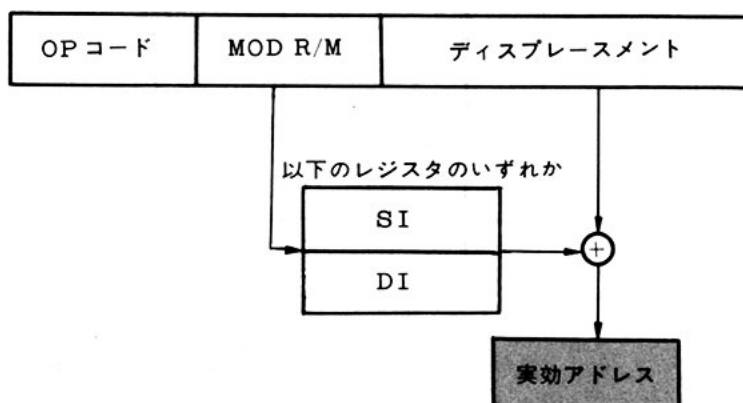
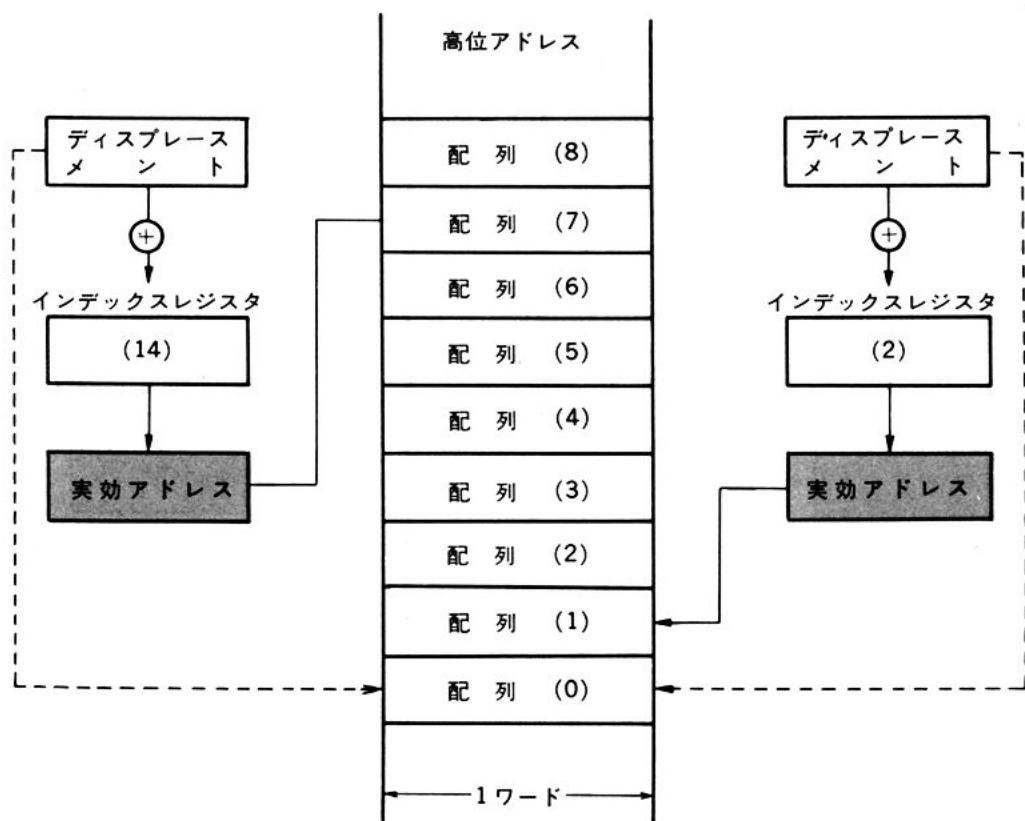



図 7・7 インデックスアドレッシングによる配列のアクセス

図 7・8 ベースを持ったインデックスアドレッシング

上位アドレス

ディスプレースメント
(6)
ベースレジスタ (BP)
インデックスレジスタ
(12)
実効アドレス

パラメータ 2
パラメータ 1
IP
(旧) BP
(旧) BX
(旧) AX
配　列　(6)
配　列　(5)
配　列　(4)
配　列　(3)
配　列　(2)
配　列　(1)
配　列　(0)
カ　ウ　ン　ト
一時ストレージ
状　態　情　報

ディスプレースメント
(6)
(BP) ベースレジスタ
インデックスレジスタ
(4)
実効アドレス

1ワード

下位アドレス

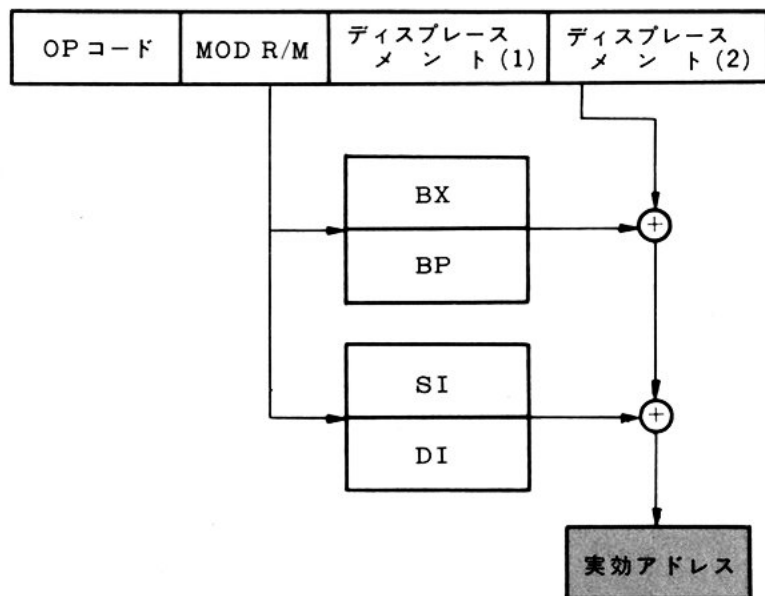
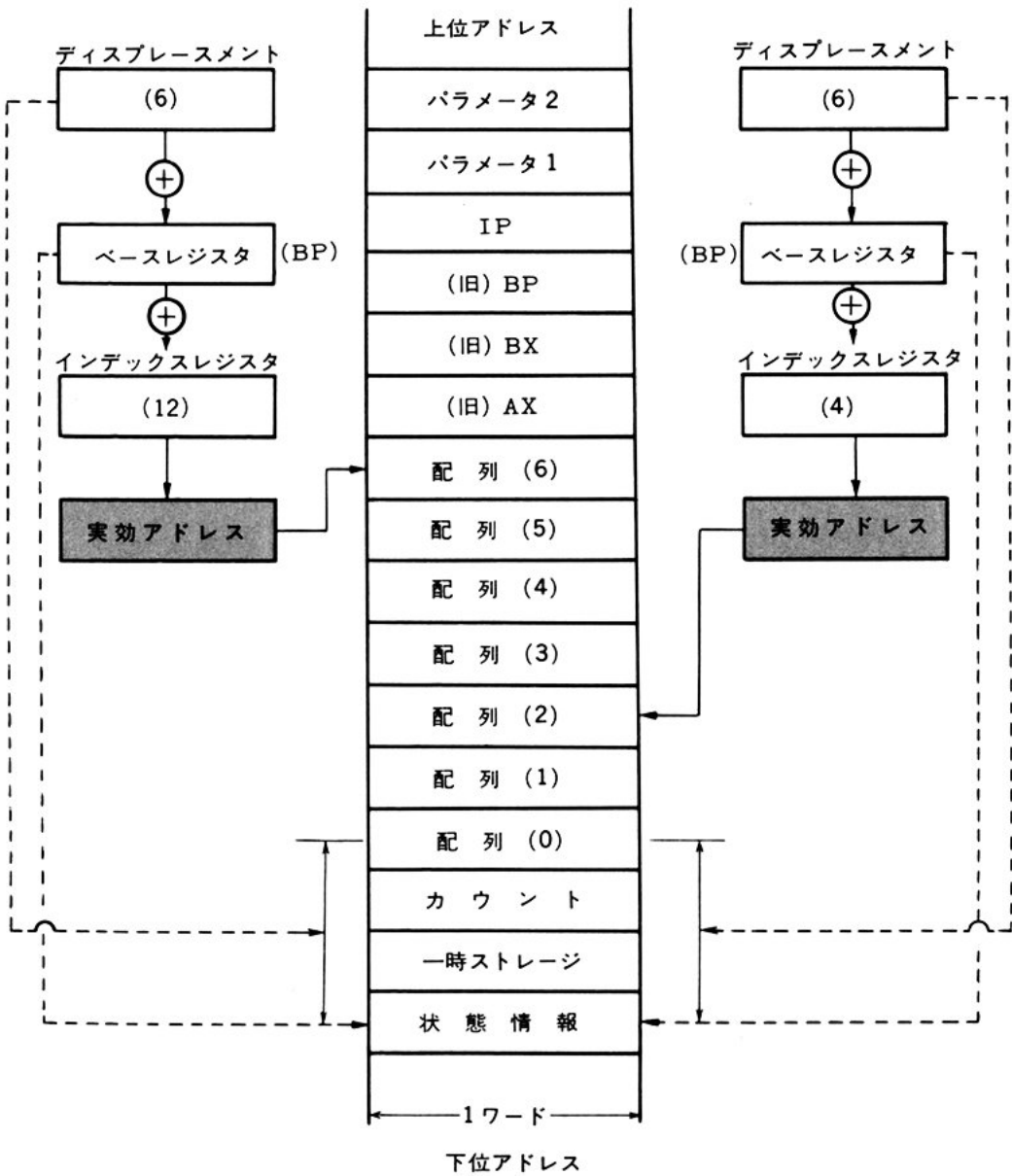



図 7・9　ベースを持ったインデックスアドレッシングによるスタック配列のアクセス

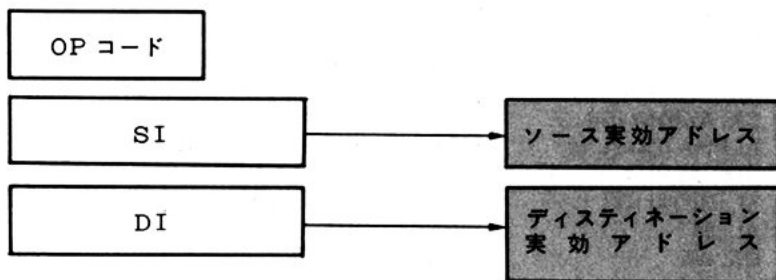



図 7・10　ストリングオペランドアドレッシング

命令中のアドレッシングモードの記述例

```
ADD   AX, BX              ; REGISTER ← REGISTER
ADD   AL, 5               ; REGISTER ← IMMEDIATE
ADD   CX, ALPHA           ; REGISTER ← MEMORY (DIRECT)
ADD   ALPHA, 6            ; MEMORY (DIRECT) ← IMMEDIATE
ADD   ALPHA, DX           ; MEMORY (DIRECT) ← REGISTER
ADD   BL, [BX]            ; REGISTER ← MEMORY (REGISTER INDIRECT)
ADD   [SI], BH            ; MEMORY (REGISTER INDIRECT) ← IMMEDIATE
ADD   [BP].ALPHA, AH      ; MEMORY (BASED) ← REGISTER
ADD   CX, ALPHA [SI]      ; REGISTER ← MEMORY (INDEXED)
ADD   ALPHA [DI+2], 10    ; MEMORY (INDEXED) ← IMMEDIATE
ADD   [BX].ALPHA [SI], AL ; MEMORY (BASED INDEXED) ← REGISTER
ADD   SI, [BP+4] [DI]     ; REGISTER ← MEMORY (BASED INDEXED)
IN    AL, 30              ; DIRECT PORT
OUT   DX, AX              ; INDIRECT PORT
```

8. システムの構成

8086 ファミリチップを使用してシステムを構成する場合のバス構成およびそのタイミングなどについて述べ，IEEE-796 バス（インテルマルチバス）との関連について記述している．また,8087（高速演算プロセッサ）および 8089（I/O プロセッサ）との組合せによるマルチプロセッサについても解説する．

8·1 8086システムの構成—ローカルバスとシステムバス

8086/8088をミニマムモードで使用する場合は従来の8ビットのCPUのようにスタンドアローンとしての使用が主になるが，マキシマムモードで使用する場合には，一つのシステムで複数のCPUに処理を分散するいわゆるマルチCPUの構成が可能になる．その複数のCPUのバス使用をコントロールするためのバスアービタ等のサポートチップ（8288/8289）により**IEEE-796バス**（インテルマルチバス）と互換性のあるシステムを構成することができる．

マルチマスタシステムのブロック図は，図8·1に示すように，二つのバス，すなわちマルチマスタローカルバスとシステムバスから成り，これらはバスコントローラおよびラッチにより分離されている．ローカルバスには8086/8088 CPUのほかに高速演算用コ・プロセッサ**8087**や，必要に応じて複数個のI/Oプロセッサ**8089**等がその内部バスだけでなく，同一のクロック，バスコントローラ，アドレスラッチおよびトランシーバも共用できる．そして，**8289バスアービタ**によってそのバスの使用が管理され，各マスタは共用のラッチおよびトランシーバを介してシステムバスに接続される．また，図8·2のように，別のI/O用ラッチ/トランシーバを設けることにより，マスタがメモリをアクセスする場合はマルチマスタシステムバスを使用し，I/Oコマンドの場合には専用のI/Oバスを使用するように構成できる．この場合，8089のようなI/Oに関係したプログラムはこの専用I/Oバスに接続され，8089によるシステムバス使用を大幅に減らし，この間システムバスは他のCPUに解放され，二つのCPUは並列動作が可能になる．

以上のシステムにさらに**8288バスコントローラ**を追加することにより，アドレス空間をシステムと常駐部分に分割でき，CPUのシステムバス使用を最小にして，それ自身のアドレス空間を走れるようになっている．このように構成することにより，他のCPUはこの常駐部分の専用メモリのアクセスはできないので，通常この部分に置かれるOSのプログラムコードおよびデータは，他のプロセッサのプログラムエラーから保護される．また，8289をさらにこれに追加することにより，この常駐バスは別のマルチマスタシステムバスになる．

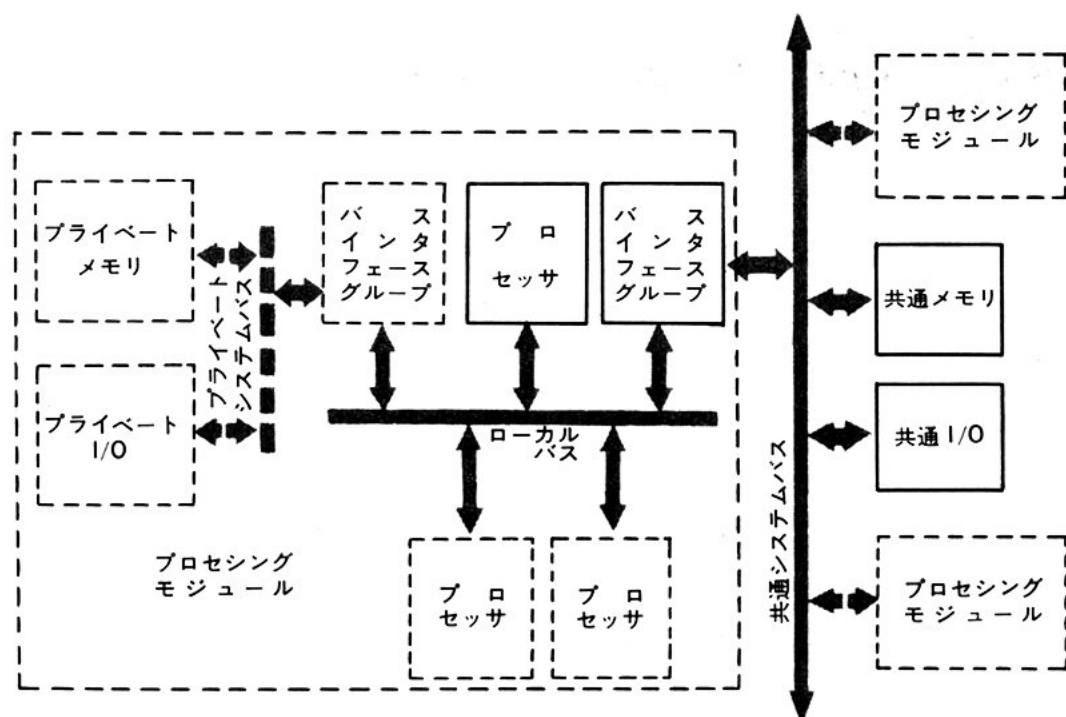



図 8・1 一般的な 8086 ファミリのバス構造

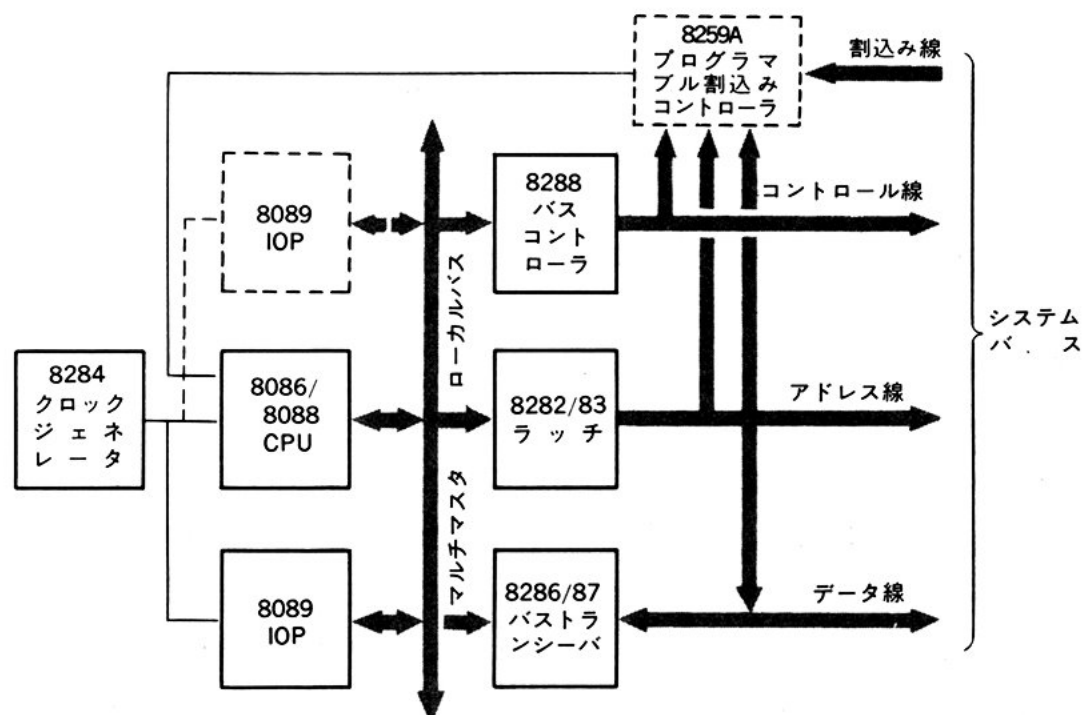



図 8・2 マルチマスタローカルバスの例

8・2 8086/8088 のバスタイミング

8086 のタイムチャートを図 8・3（リードサイクル）と図 8・4（ライトサイクル）に示す．標準の 8086 は 5 MHz のクロックで動作し，1 バスサイクルは T_1 ～ T_4 の四つのステートから成っている．リードサイクルの場合の動作は，まず，T_1 の始めからアドレス AD_0～AD_{15} および A_{16} ～ A_{19} が出力され，それと同時に ALE（アドレスラッチイネーブル）が出され，これらのアドレスを外付けのラッチ回路にラッチする．アドレスの A_0 ～ A_{15} はデータと，また A_{16} ～ A_{19} は状態情報（S_3 ～ S_7）と共用で，時間的に切り換えて使用しているのでこのラッチが必要となる．また，そのバスサイクルがメモリに関係したものか，I/O 関連の命令かを示す $M/\overline{IO}$（8088 の場合は $IO/\overline{M}$）という信号を全バスサイクルにわたり出力し，後から出される $\overline{RD}$ 信号との組合せで，メモリリード $\overline{MEMR}$ または I/O リード $\overline{IOR}$ の信号を作成する．

また，システムを構成する双方向性バスバッファ（8286/8287）のデータ方向（リード動作/ライト動作）切換えのための $DT/\overline{R}$ と，リード/ライト動作時だけバスに接続するようコントロールするための信号 $\overline{DEN}$（データイネーブル）が供給されており，システムの構成を容易にしている．リード動作の場合は，以上の信号が揃い，$\overline{RD}$ が出ると選ばれたメモリまたは I/O のデータが AD_0～AD_{15} 上に現れ，CPU に読み取られ，その後 T_4 ステートでそのバスサイクルを完了する．データバスは T_2 ステートの中間から T_3 ステートにかけて，読込み用のバス切換えのために，フローティングの状態が存在する．

ライトサイクルでもほぼ同じ．ただ $DT/\overline{R}$ の極性がリードの場合の逆になり，$\overline{DEN}$ がライトタイミングのマージンのため広くなる．以上の信号とアドレスのラッチ後，$\overline{WR}$ 信号が出てバス上のデータが選ばれたメモリまたは I/O に書き込まれる．8088 の場合は，データバスが 8 ビットであることから，アドレスと共用になるのは AD_0～AD_7 までで，あとの A_8～A_{15} は全バスサイクル中連続して出力されるのでラッチの必要はない．また，注意を要するのは，メモリと I/O 動作の切換え信号が 8088 の場合は $IO/\overline{M}$ になり，8086 とは逆である．

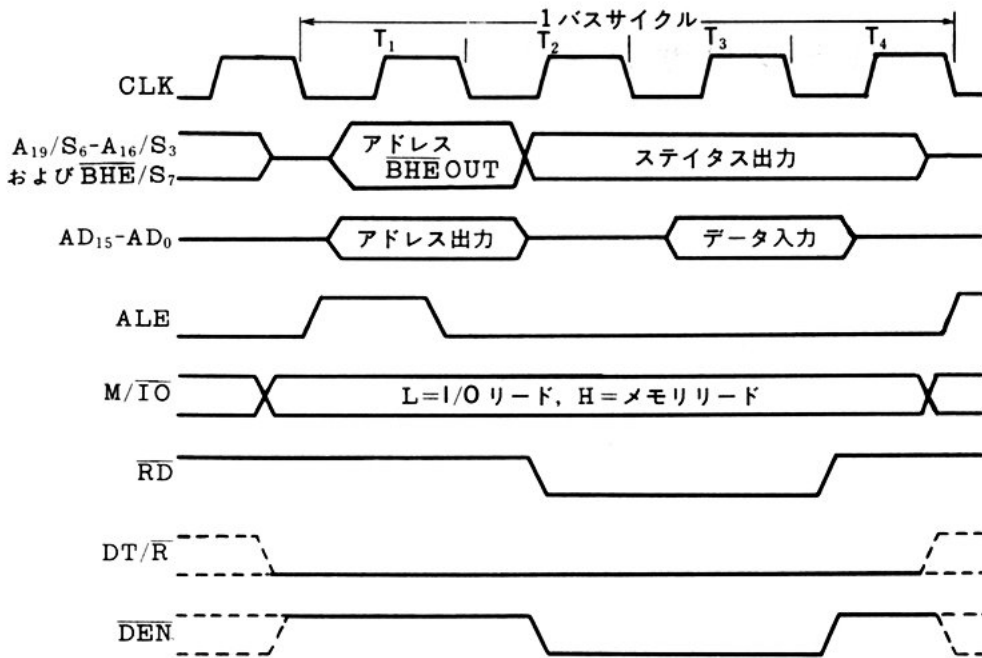



図 8・3 8086 の READ バスサイクル

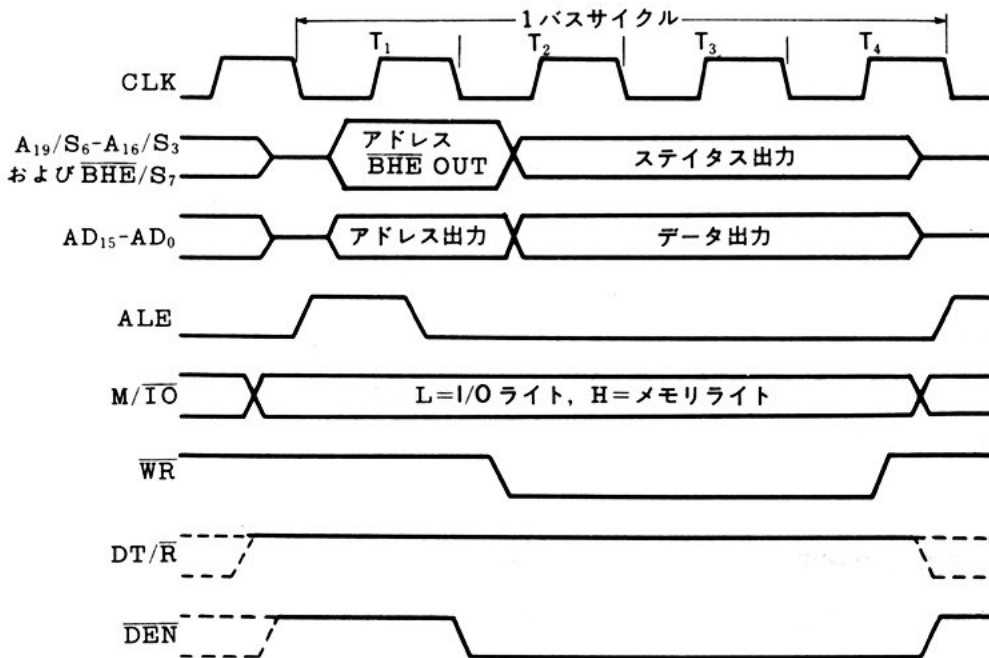



図 8・4 8086WRITE バスサイクル

8·3　マルチプロセシング

マイクロコンピュータの価格が急激に下がってきたので，従来は一つのCPUが全システムの処理をすべて行っていたものを，複数のCPUを使用してその仕事を分担させるようになってきた．この場合，各CPUはデータの相互の受渡しや，共通バスの使用時などのほかは独立して自分に分担された仕事を並列処理できるので，システム全体のスループットは大幅に向上する．図8·5のマルチプロセッサシステムではI/O関係の処理はすべて8089 IOPに任せ，8086はシステム全体の管理と演算処理等を担当する．8086/8088では，このマルチプロセシングのための考慮が，ハード/ソフト面から行われている．

次に，このマルチプロセシングで使用されるいくつかの機能について述べる．

〔1〕 **リクエスト/グラント**（$\overline{RQ}/\overline{GT}$）**機能**　8086/8088をマキシマムモードで使用すると，ミニマムモードの場合のHOLD/HOLDAに相当する信号として，2チャネルの$\overline{RQ}/\overline{GT}$という端子が用意されており，複数のCPUによるローカルバスの共用を可能にする．この$\overline{RQ}/\overline{GT}$端子は双方向性の制御線になっており，ハンドシェーク動作により，リクエスト(要求)，グラント(許可)およびリリース(解放)の三つのシーケンスを実行する．まずバス使用を要求するプロセッサが$\overline{RQ}/\overline{GT}$線にパルスを送ると，メインCPUはHOLD状態に入ったことを示すためのパルスを同一線上に返し，バスを解放する．BIUはこの期間バスから切り離されるが，EUはその後バスの使用を要求する命令が出るまで今までの仕事の実行を継続する．最後にその要求中のプロセッサのバス使用が終了すると，それを知らせるためのパルスをメインCPUに送り，メインCPUは再びバスの使用権を得る．$\overline{RQ}/\overline{GT_0}$は$\overline{RQ}/\overline{GT_1}$より優先度が高く，同時に起こった場合は0のほうが優先される．

〔2〕 **バスロック機能**　バスロックは8289バスアービタとともに使用され，命令の前に付加される**ロックプリフィックス**という1バイトの命令により，その命令の実行中はそのシステムバスの使用を保証するというものである．EUがロックプリフィックスをデコードすると，BIUにその後のクロックサイクルの間$\overline{LOCK}$

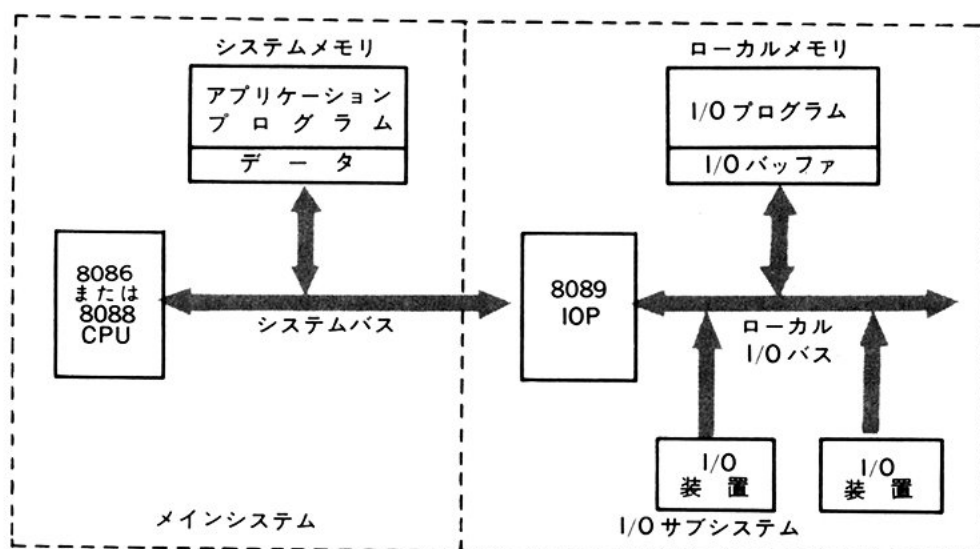



図 8・5 8086/8089 マルチプロセッサシステム

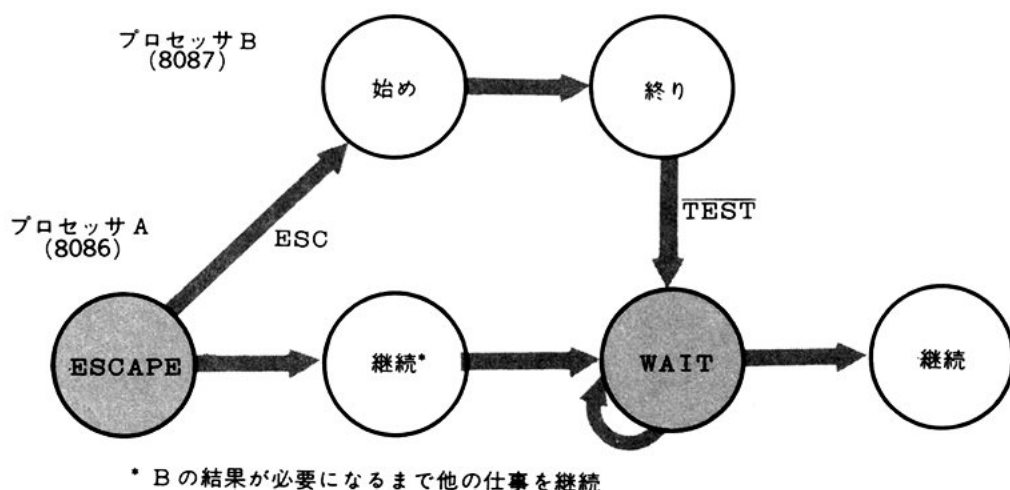



図 8・6 マルチプロセッサシステムにおける ESC の使用

信号を出すよう知らせ，この信号はその命令の実行が終了するまでアクティブになっており，他のCPUのバスの使用を禁止する．この機能はソフトウェアコントロールが可能である．

〔3〕 WAIT/$\overline{\text{TEST}}$　WAIT命令と$\overline{\text{TEST}}$信号は対になっており，EUがWAIT命令を実行したときに，$\overline{\text{TEST}}$端子がアクティブでない（HIGH）の場合は，CPUはアイドル状態となり5クロック間隔でその$\overline{\text{TEST}}$端子をテストする．$\overline{\text{TEST}}$端子がアクティブ（LOW）になると次の命令に実行を移す．これは$\overline{\text{TEST}}$信号を使い外部事象との同期を可能にする．

〔4〕 **エスケープ機能**　ESC命令は8087のようなコ・プロセッサが，8086のプログラムから命令やオペランドを受け取ることを可能にする．前述のWAIT/$\overline{\text{TEST}}$と組み合わせて，Aプロセッサの出したESC命令をBプロセッサが受け取り，要求された演算ルーチンを起動する．プロセッサAはそのまま後の処理を続行し，結果が必要になるとWAIT命令を実行し，TEST端子をチェックし，それが上がってくると，その結果の準備ができたということで，メモリ中にあるその演算結果をもらって，後の処理を続行する．

8・4 マルチバスのアーキテクチャ

マルチバスはインテル社の1ボードコンピュータ SBC シリーズで採用されているマイクロコンピュータ用の標準バスで，**IEEE-796** として広く使用されている．8086 をマキシマムモードとし，8288 バスコントローラと 8289 バスアービタを組み合わせるとマルチバスと互換性のあるシステム構成が可能になる．マルチバス上の各モジュールはマスタまたはスレーブとして設計されており，通常マスタがバスの使用権を持ち，データの転送等を始動し，スレーブはその動作の対象となる．マルチバスは 8 ビットまたは 16 ビットのマスタ/スレーブの混在が可能で，16 本の双方向性データバス，20 本のアドレス線，8 本の割込み線および数本のコントロール線より構成される．マルチバスの信号の一覧表は付録を参照されたい．

マルチバスではデータバスを8ビット/16 ビット両方に使用でき，図 8・7 に示すように，16 ビットの場合の高位奇数アドレスバイトをデータバスの下位 8 ビットに切り換えるための回路が付加されている．

共通の割込み線は $INT_{0\sim7}$ の 8 本があり，スレーブ CPU からの割込み要求をワイヤードオアして，マスタ CPU ボード上のプログラマブル割込みコントローラ (8259A) の割込み要求ラインに接続する．また，割込みコントロールの他の構成法として，スレーブ CPU にスレーブの 8259A が搭載されている場合があり，データバスを通じて割込みベクタアドレスをマスタ CPU へ伝送する．この場合の動作は 8259A のマスタ/スレーブ動作と同じである．

次に，一つのシステムに複数のマスタが存在するいわゆるマルチマスタ動作の場合は，その各マスタのマルチバスを使用する優先準位の決定方法には直列優先方式と並列優先方式の二つがある．直列優先方式は，いわゆるディジィチェーン接続で，図 8・8 のようにアクティブローの二つの信号 $\overline{\text{BPRN}}$（バスプライオリティイン）と $\overline{\text{BPRO}}$（バスプライオリティアウト）により，優先度の最も高いものの $\overline{\text{BPRN}}$ を GND に接続し，以下は接続順序により順位が決まる．優先準位の高いものがバスを使用中は，その $\overline{\text{BPRO}}$ 出力をハイとして，それ以下のもののバ

ス使用を禁止する．並列優先方式は，各マスタからの要求を優先エンコーダ/デコーダによりその順位を判定し，要求中の最高位のものの $\overline{\mathrm{BPRN}}$ をローにする．

ユーザバス
マルチバス
下位バイトバッファ
8287
D_0-D_7
DAT_0/-DAT_7/
方向切換
スワップバイトバッファ
DAT_0/-DAT_F/
D_8-D_F
DAT_8/-DAT_F/
上位バイトバッファ
バッファ
74S00
スワップBYTE/
BHEN/
ADR_0
AEN/
74S32
74S04

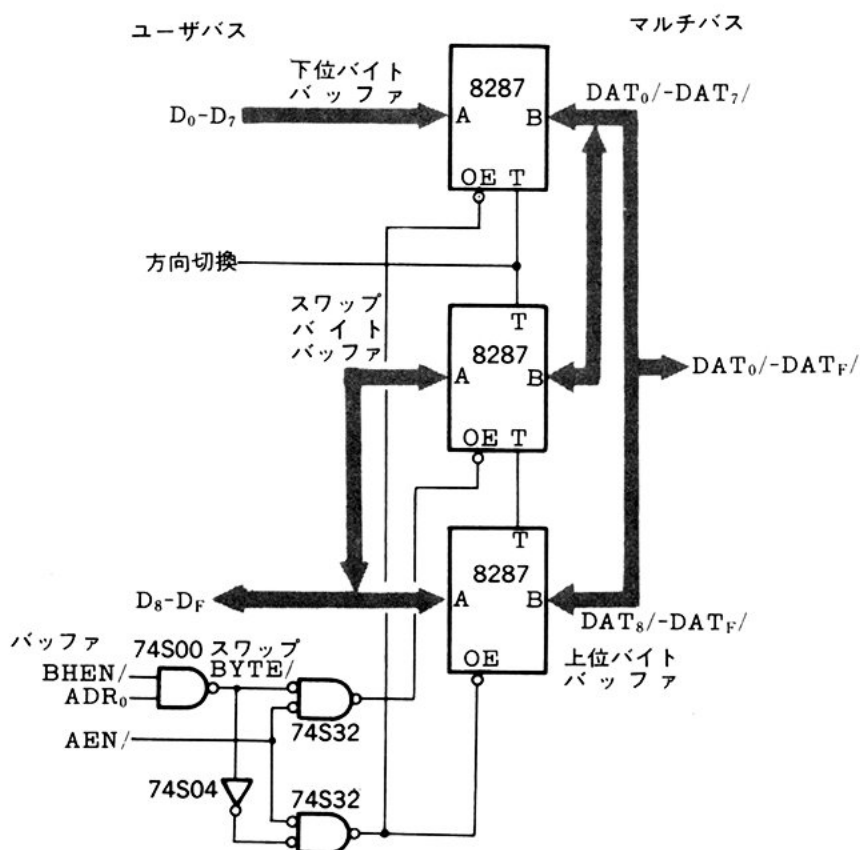



図 8・7　8/16 ビットバス切換えドライバ回路

図 8・8　直列優先接続

(最高位)NO.1
優先
BPRN/
BREQ/
NO.2
優先
BPRN/
BREQ/
NO.7
優先
BPRN/
BREQ/
NO.8(最下位)
優先
BPRN/
BREQ/
バス優先度
リゾルバ
優先エンコーダ
74148
優先デコーダ
74S138
その他のマスタ入力
その他のマスタ出力
7
6
5
4
3
2
1
0

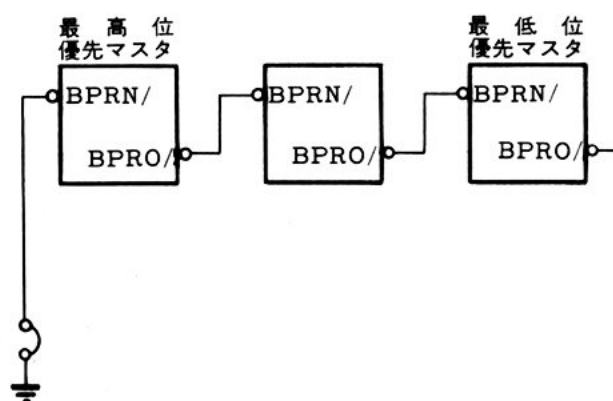
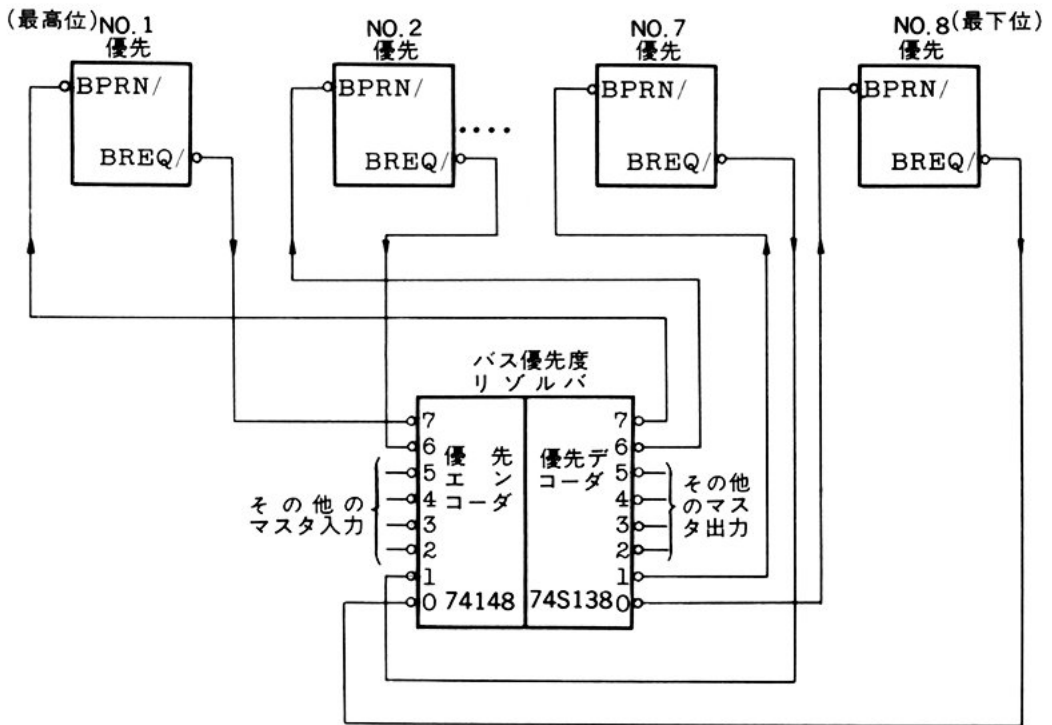



図 8・9　並列優先接続

16ビット CPU になるとそれに接続されるI/O装置も，各種ターミナルやコントローラ，フロッピーディスク/ハードディスクコントローラなど多種多様になってくるので，ソフトウェア体系も含めてそのI/Oシステムを階層構造の構成にするのが望ましい．

たとえば，ユーザのアプリケーションモジュールがディスクファイルを読む場合に，そのファイルがディスク上のどこに位置しているかとか，そのディスクセクタのサイズがどれだけであるかなどを知る必要はなく，単にそのファイルシステムプロセデュア（サブルーチン）をコールすることにより，一連のファイルに関係したコマンド（OPEN，CLOSE，READ，WRITE）を実行する．

そのファイルシステムは，次にその下のレベルのI/Oスーパーバイザ中のプロセデュアをコールし，このI/Oスーパーバイザが最初のアプリケーションモジュール中のI/O要求と，最低位のモジュール（I/Oドライバ）によりコントロールされるI/O装置との間の橋渡し役を行う．

このスーパーバイザでは，異なった種類の装置のためには別々のモジュールが用意され，またその装置に依存するコードもそのモジュール中に含まれる．また，実際に最終のI/O装置のコントロールを行うモジュールは最低位のレベルで，その装置特有のコントロールプログラムを含んでおり，この部分は個々の装置に対して用意されなければならないプログラムである．

9. 周辺ファミリチップ

8086のシステムを構成する周辺ファミリチップのおのおのについて記述している．ここでは特に8086用に開発されたものだけを掲げ，従来の80用のものは載せていないが，タイミングなど使用法は特に変わっているところはなく，そのまま使用できる．ただし，ものによってはウエイトの必要なものもあるので注意を要する．

9・1 クロックジェネレータ（8284A）

クロックジェネレータ8284Aは8086/8088/8089用の基準クロックの発振回路で，外付けの水晶振動子または外部クロックに同期したクロックを発生する．水晶発振回路の出力は内部で1/3に分周されCPUのクロックになるので，5MHz動作の場合には15MHzの水晶振動子が使用される．

また，同じシステム中で使用される周辺チップに供給するためのクロックは，このCPUに供給するクロックをさらに1/2に分周して使用している．

図9・1に示すブロック図でEFIは外部クロックで，$F/\overline{C}$により内部クロックとの切換えを行い，どちらかを選択する．また，1システム中でいくつかのクロックを使用する場合に，それらを同期させるためのCSYNCという端子があり，他のクロックとの同期がとれるようになっている．

その他の付属回路としてシュミットトリガ回路内蔵のシステムリセット回路があり，CPUだけでなく，システム全体のために，クロックの復縁に同期したリセット信号を発生する．

また，二つのマルチマスタシステムバスを適合させるための二つのREADY信号（RDY_1およびRDY_2）があり，それぞれ$\overline{AE_1}$，$\overline{AE_2}$でゲートされている．このREADY入力（CPUへの）の立上りはRDYのセットアップとホールド時間の規格を満足させるためにクロックに同期させる必要がある．READYの立下りはクロックに同期している必要はないが，確実なシステム動作という観点からは同期していることが望ましい．8284AのREADY回路は，これらの要求を満足している．

また，発振回路では，オーバトーン用クリスタルによる発振用のTNK（タンク回路）端子があり，外付けのLCタンク回路により第3高調波による発振が可能になっているが，通常はこのような使用はしない．

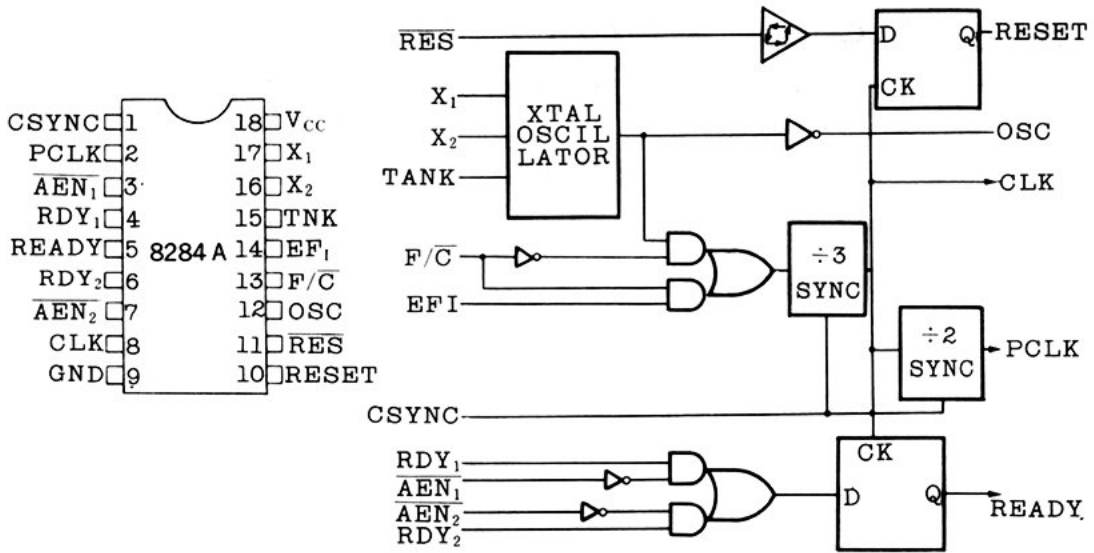



図 9・1 8284 Aクロックジェネレータ

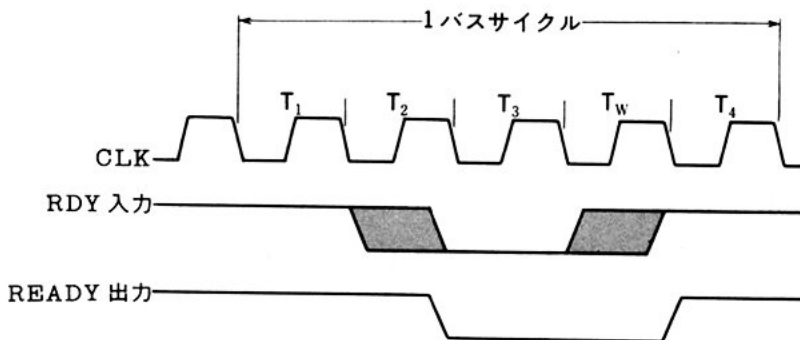



図 9・2 WAIT タイミング

9・2 8ビットラッチバッファと 8ビット双方向性バストランシーバ

〔1〕 **8ビットラッチバッファ**（8282/8283） 8086/8088のアドレスのラッチおよびバッファ用として8ビットラッチバッファ8282/8283がある．8282はノンインバーティング，8283はインバーティングで，ともに3ステート出力になっている．

動作はSTB信号がHIGHになるとDI_0～DI_7の入力がラッチ内に入り，HIGHからLOWへの立下りで，ラッチされる．通常，DI_0～DI_7はアドレスのAD_0～AD_{15}（8088の場合はAD_7）またはA_{16}～A_{19}で，STB信号としてはALEを使用し，アドレスのラッチを行う．$\overline{OE}$は出力イネーブルのコントロール端子でLOWのときにラッチされているデータが出力D_0～D_7に現れる．直流特性としてはLOWの場合のシンク電流は32mA，HIGHの場合の出力電流は最大で－5mAである．

〔2〕 **8ビット双方向性バストランシーバ**（8286/8287） 8086/8088のマルチプレクスされたバスのドライブ能力はシンク電流2mAおよび負荷容量100pFまでと規定されており，これは通常2～3の周辺チップと2～4個のメモリチップの構成でこの限度となるので，かなり小さいミニマムシステムの構成以外は，データバスに双方向性のバスバッファを挿入しなければならない．この用途のための8ビットのトランシーバとしてノンインバーティングの8286およびインバーティングの8287がある．

動作はバスの方向を切り換えるためのT端子があり，これがHIGHのときはデータがA→Bへ，LOWのときはB→Aの方向となる．ここの端子には8086/8088から供給される$DT/\overline{R}$の信号を接続する．そして，出力コントロールのための$\overline{OE}$端子があり，これがLOWのとき，このバッファはアクティブとなり，HIGHになると3ステートとなる．この端子は通常CPUから供給される$\overline{DEN}$（データイネーブル）を接続する．これらのバッファを挿入することによりシステムバス側は32mA/300pFまで拡張され，CPUとのインタフェース側は10mA/100pFまでドライブが可能になる．

図 9・3　8282/8283 オクタルラッチ

A0 1 / 20 VCC
A1 2 / 19 B0
A2 3 / 18 B1
A3 4 / 17 B2
A4 5 / 16 B3
A5 6 / 15 B4
A6 7 / 14 B5
A7 8 / 13 B6
OE 9 / 12 B7
GND 10 / 11 T
8286

A0 1 / 20 VCC
A1 2 / 19 B0
A2 3 / 18 B1
A3 4 / 17 B2
A4 5 / 16 B3
A5 6 / 15 B4
A6 7 / 14 B5
A7 8 / 13 B6
OE 9 / 12 B7
GND 10 / 11 T
8287

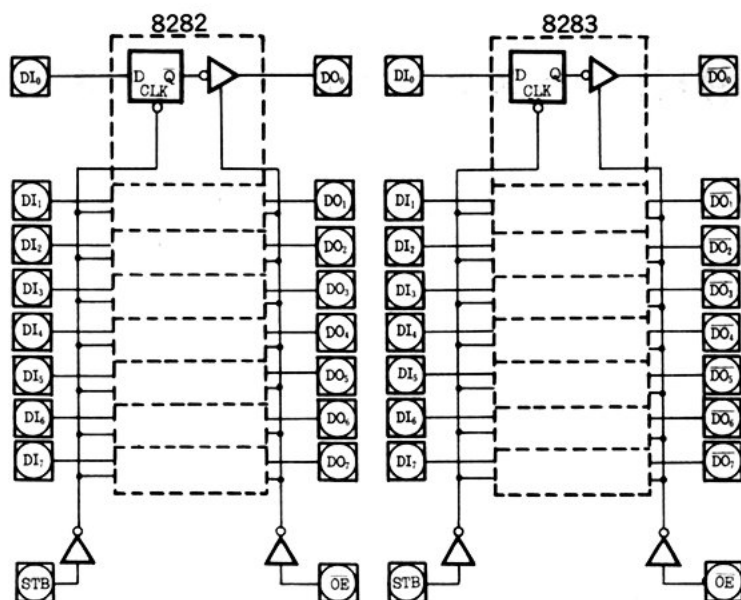
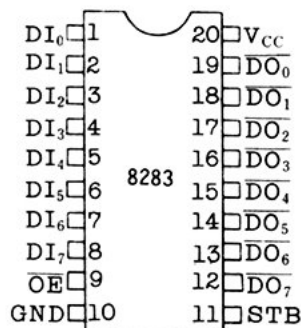
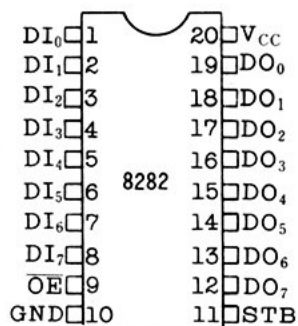
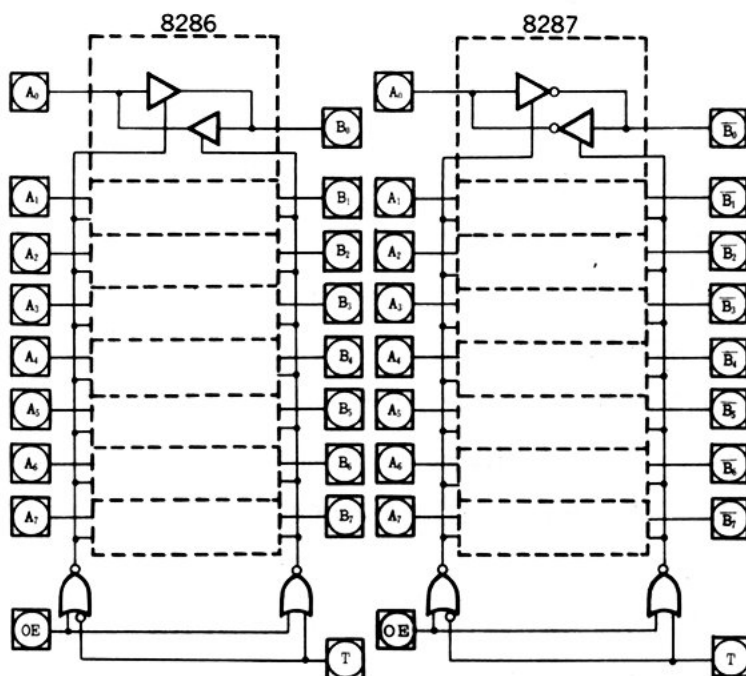
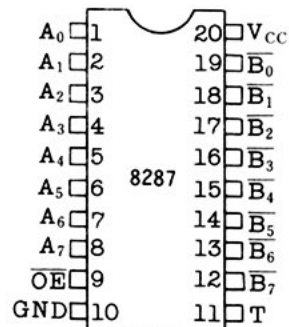
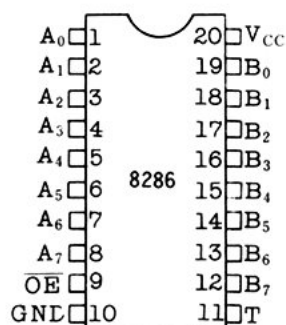



図 9・4　8286/8287 オクタルバストランシーバ

9・3 バスコントローラとバスアービタ

〔1〕 **バスコントローラ**（8288）　8086/8088をマキシマムモードで使用する場合，CPUはコントロール信号を直接出力せずに状態信号 $\overline{S_0}$～$\overline{S_2}$ として出力し，8288がこれを受け取りデコードすることによってミニマムモードの信号に相当するコントロール信号が得られる．8288の場合は $\overline{\text{MRDC}}$/$\overline{\text{MWTC}}$，$\overline{\text{IORC}}$/$\overline{\text{IOWC}}$ の信号が個々に出力され*，さらに通常より1クロック分早めのコントロール信号 $\overline{\text{AMWC}}$/$\overline{\text{AIOWC}}$ が出力されてアクセスタイムの改善に役立っている．

CEN（コマンドイネーブル）がLOWの場合は，すべてのコントロール信号は無効で，HIGHでアクティブとなる．IOBとMCE/$\overline{\text{PDEN}}$は対の信号で，IOBがLOWの場合は，8259Aの割込みシーケンスの間にマスタ割込みコントローラからカスケードアドレスを読み出すのに使われる．IOBがHIGHの場合は，I/O命令の間データバストランシーバをイネーブルにする信号として使用される．

〔2〕 **バスアービタ**（8289）　8288バスコントローラと組み合わせて8086/8088/8089をマルチマスタシステムバスにインタフェースするのに使われ，多数のマスタCPUがシステムバスを共用できる．プロセッサはそのバスアービタがシステムバス使用権を獲得するまではWAIT状態で，獲得後そのバスコントローラ，トランシーバ，アドレスラッチ等をイネーブルにする．データの転送が開始されると，アクノレージ信号XACKがスレーブ装置からCPUに返される．

バス使用優先度の決定方式にはｉ）並列優先方式，ii）直列優先方式，およびiii）回転優先方式の三つがある．並列の場合は図8・9のように，各アービタからのバスリクエスト $\overline{\text{BREQ}}$ をエンコードし，その要求情報からその中で一番優先度の高いものを選ぶ（$\overline{\text{BREQ}}$＝LOW）．直列の場合は各アービタがディジーチェーンに接続されており，高位優先度のものがバスを使用中の場合はその $\overline{\text{BPRN}}$ 出力をHIGHにして以下のものを禁止する．$\overline{\text{BPRN}}$＝LOWの場合に，$\overline{\text{BREQ}}$ がある場合は，そのアービタがバスの使用権を得る．次のもののリクエストがない場合は，その $\overline{\text{BPRO}}$ をLOWにして次々に下位へ渡してゆく．

---

* ミニマムモードの場合は $\overline{\text{RD}}$，$\overline{\text{WR}}$ と $\overline{\text{IO}}$/M（8088は IO/$\overline{\text{M}}$）

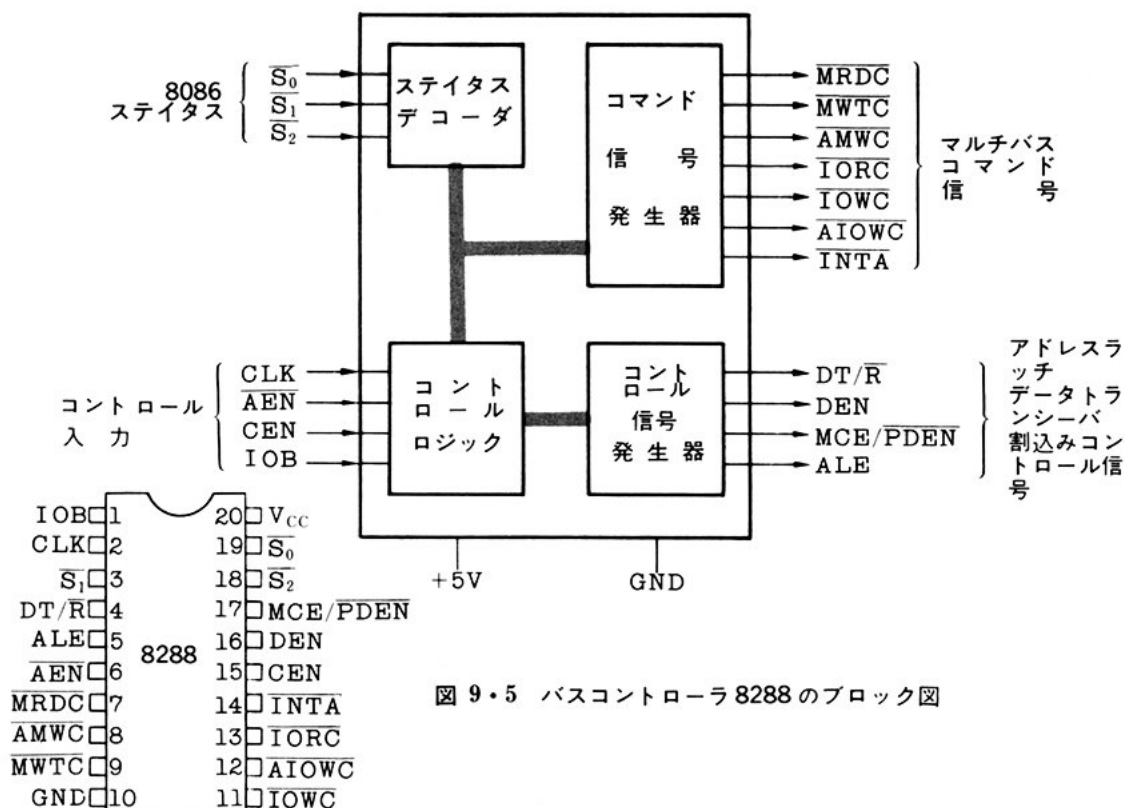



図 9・5　バスコントローラ 8288 のブロック図

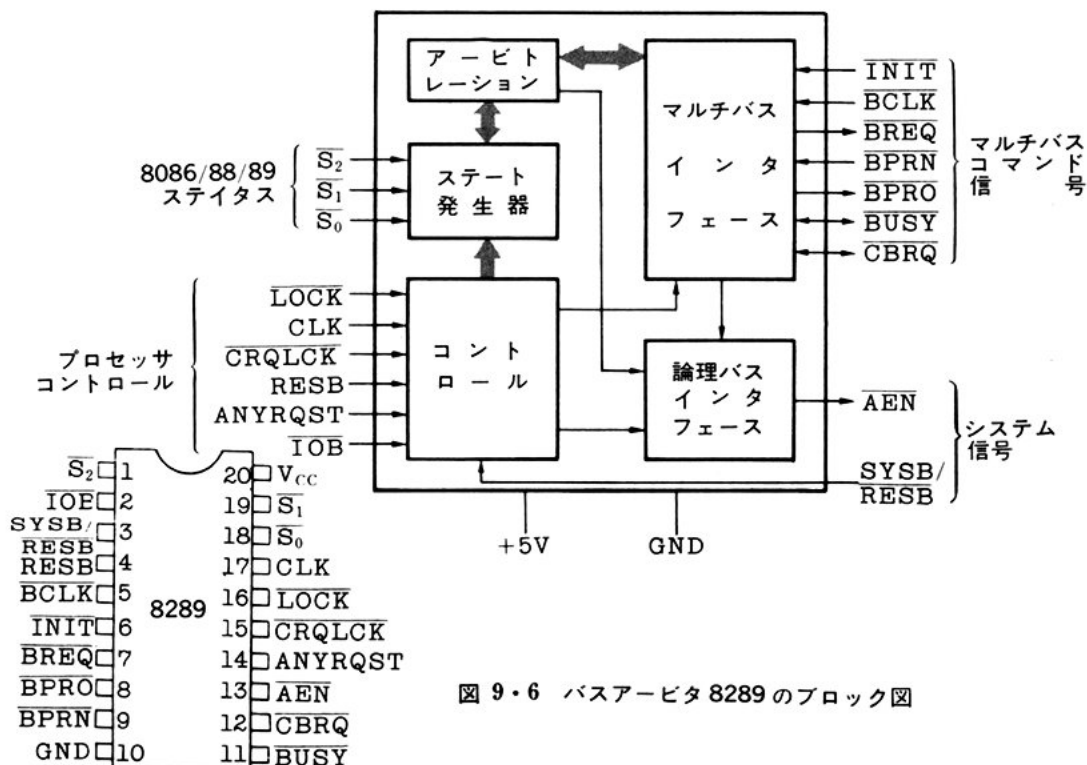



図 9・6　バスアービタ 8289 のブロック図

9・4 割込みコントローラ（8259A）

8086の割込みは，5章に述べた割込みポインタテーブルと8259Aが一体となって，256レベルのベクトル化された割込みとなっている．8259Aは8080/85モードと8086モードの二つの動作モードがあり，コントロールワードの設定によりいずれかの動作を選択する．この二つは割込みに対する応答のシーケンスが異なるが，ここでは8086モードについてだけ述べる．

8259Aはプログラムからの設定により種々の動作モード/機能が選べるようになっており，システムの電源投入またはリセットに続き，それらのコントロールワードレジスタの初期設定を行う．このコントロールワードには初期化コマンドワードICW（イニシャライゼーション コマンド ワード）と動作コマンドワードOCW（オペレーション コマンド ワード）の2種類があり，さらにICWとして4種類，OCWとして3種類のコマンドがある．

上記の設定が完了すると8259Aは割込み要求の受付けが可能になり，次のようなシーケンスで割込みの処理が行われる．

（1） 割込み要求線 IR_0～IR_7 に割込み要求がくると，その対応する割込み要求レジスタIRRのビットをセットする．複数の割込みの同時受付け可．

（2） 8259Aはこれらの要求を評価し，CPUへ割込み要求INTを送る．

（3） CPUはINTに対する応答信号として $\overline{INTA}$ パルスを返してくる．

（4） CPUからの $\overline{INTA}$ を受け取ると，IRRの中の最も優先度の高い割込みに対応するビットがISRにセットされ，そのIRRの対応するビットは，受付け完了ということからリセットされる．

（5） 8086CPUは2番目の $\overline{INTA}$ パルスを出力し，その期間に8259Aはデータバス上に8ビットのポインタ情報を落とし，CPUはそれを読み込む．

（6） これで割込みサイクル完了であるが，ICW_4 の中でAEOI（オートマチック エンド オブ インタラプト）モードが選択されている場合は，2番目の $\overline{INTA}$ の終りでISRの対応するビットがリセットされる．さもないと，割込み処理ルーチンの終りでEOIコマンドが出るまで，ISRはそのままになる．

8259A

$\overline{CS}$	1	28	V_{CC}
$\overline{WR}$	2	27	A_0
$\overline{RD}$	3	26	$\overline{INTA}$
D_7	4	25	IR_7
D_6	5	24	IR_6
D_5	6	23	IR_5
D_4	7	22	IR_4
D_3	8	21	IR_3
D_2	9	20	IR_2
D_1	10	19	IR_1
D_0	11	18	IR_0
CAS_0	12	17	INT
CAS_1	13	16	$\overline{SP}/\overline{EN}$
GND	14	15	CAS_2

ピン名称

D_7-D_0	データバス（双方向性）
$\overline{RD}$	リードコントロール
$\overline{WR}$	ライトコントロール
A_0	コマンド選択アドレス
$\overline{CS}$	チップ選択
CAS_2-CAS_0	カスケードライン
$\overline{SP}$	スレーブプログラム入力
INT	割込み出力
$\overline{INTA}$	割込みアクノレージ入力
IR_0-IR_7	割込み要求ライン

$\overline{INTA}$　INT

D_7-D_0　データバスバッファ

コントロールロジック

$\overline{RD}$　$\overline{WR}$　A_0　リード/ライトロジック　$\overline{CS}$

インサービスレジスタ(ISR)　優先度判定　割込み要求レジスタ(IRR)　IR_0 IR_1 IR_2 IR_3 IR_4 IR_5 IR_6 IR_7

CAS_0　CAS_1　CAS_2　カスケードバッファ/比較器　$\overline{SP}/\overline{EN}$

割込みマスクレジスタ(IMR)

内部バス

図 9・7　割込みコントローラ 8259A のピン構成とブロック図

図 9・8　8259A のカスケード接続（スレーブ最大８個まで）

8253および8259Aの初期化プログラム例

```
INIT                PROC        NEAR
; THIS PROCEDURE IS CALLED FOR BOTH WARM AND COLD STARTS TO INITIALIZE
;     THE 8253 AND THE 8259A. THIS ROUTINE DOES NOT USE STACK, DATA, OR
;     EXTRA SEGMENTS, AS THEY ARE NOT SET PREDICTABLY DURING A WARM START.
;     INTERRUPTS ARE DISABLED BY VIRTUE OF THE SYSTEM RESET.

; INITIALIZE 8253 COUNTER 1 - OTHER COUNTERS NOT USED.
; CLK INPUT TO COUNTER IS ASSUMED TO BE 1.23 MHZ.

LO50MS              EQU         000H            ; COUNT VALUE IS
HI50MS              EQU         0F0H            ;     61440 DECIMAL.
CONTROL             EQU         0D6H            ; CONTROL PORT ADDRESS
COUNT_1             EQU         0D2H            ; COUNTER 1 ADDRESS
MODE2               EQU         01110100B       ; MODE 2, BINARY

                    MOV         DX,CONTROL      ; LOAD CONTROL BYTE
                    MOV         AL,MODE2
                    OUT         DX,AL
                    MOV         DX,COUNT_1      ; LOAD 50MS DOWNCOUNT
                    MOV         AL,LO50MS
                    OUT         DX,AL
                    MOV         AL,HI50MS
                    OUT         DX,AL
                    ; COUNTER NOW RUNNING, INTERRUPTS STILL DISABLED.

; INITIALIZE 8259A TO: SINGLE INTERRUPT CONTROLLER, EDGE-TRIGGERED,
;     INTERRUPT TYPES 32-40 (DECIMAL) TO BE SENT TO CPU FOR INTERRUPT
;     REQUESTS 0-7 RESPECTIVELY, 8086 MODE, NON-AUTOMATIC END-OF-INTERRUPT.
;     MASK OFF UNUSED INTERRUPT REQUEST LINES.

ICW1                EQU         00010011B       ; EDGE-TRIGGERED, SINGLE 8259A, ICW4 REQUIRED.
ICW2                EQU         00100000B       ; TYPE 20H, 32 - 40D
ICW4                EQU         00000001B       ; 8086 MODE, NORMAL EOI
OCW1                EQU         11110111B       ; MASK ALL BUT IR3
PORT_A              EQU         0C0H            ; ICW1 WRITTEN HERE
PORT_B              EQU         0C2H            ; OTHER ICW'S WRITTEN HERE

                    MOV         DX,PORT_A       ; WRITE 1ST ICW
                    MOV         AL,ICW1
                    OUT         DX,AL
                    MOV         DX,PORT_B       ; WRITE 2ND ICW
                    MOV         AL,ICW2
                    OUT         DX,AL
                    MOV         AL,ICW4         ; WRITE 4TH ICW
                    OUT         DX,AL
                    MOV         AL,OCW1         ; MASK UNUSED IR'S
                    OUT         DX,AL
; INITIALIZATION COMPLETE, INTERRUPTS STILL DISABLED
                    RET
INIT                ENDP
```

10. 8086/8088システムの開発装置

8086システムの開発装置およびソフトウェアのサポート体系について述べ，インサーキットエミュレータ（ICE86）および評価用ボード（SDK86）についても紹介する．また，8086搭載の1ボードコンピュータの一例としてSBC86/12Aを示している．

10・1 インテルMDSマイクロコンピュータ開発装置

8086/8088システムの開発装置としてはインテル社のMDSシリーズの開発装置があり，旧形の**MDS800**および**MDSシリーズII**は8080/8085ベースの開発装置で，**クロスアセンブラASM86**および**クロスコンパイラPL/M-86**のほか，**リンカーLINK86**および**ロケータLOC86**等が提供されている．また8086をベースにした**MDSシリーズIII**は7.3Mバイトのハードディスク付きでシリーズIIと比較して 3～5 倍の処理速度の向上が図れる．

8086の開発には，システムプログラムコードや開発するプログラムのサイズの増大から，**MDSシリーズIIモデル230**(フロッピーディスク容量2.25Mバイト）以上のものを使用することが望ましい．MDS用のオペレーティングシステム（OS）としては**ISIS-II**（Intel Software Implemented Supervisor）があり，ディスケットファイルのフォーマッティングのほか，ファイルのコピー，削除などのファイル管理の機能と，DEBUGコマンドのように簡単なモニタプログラムの下でブレークポイントを設定してプログラムのデバックを行うことも可能である．さらに詳細な解析のためには，次に述べる**インサーキットエミュレータ**(**ICE86**)が必要になる．ISIS-IIのもとでサポートされている高級言語としては，PL/M-86以外に**FORTRAN86**および**PASCAL86**も提供されており，ユーザの必要に応じて選択することができる．

また，ソースプログラムの編集用のラインエディタ（EDIT）とスクリーンエディタのCREDITがあり，後者は修正箇所までカーソルを移動し簡単に修正ができるのが特徴である．

アセンブラまたはコンパイラで作成したオブジェクトプログラムは，プログラムロード時に実際のロケーションが決定できるように，リロケータブルになっており，最終段階で他のサブルーチンやライブラリプログラムとLINK86によりリンクして1本のプログラムにした後，ロケータLOC86により実際のメモリロケーションを決定し，実行可能なアブソルートなオブジェクトコードを作成する．また，これをPROMに書く場合に，プログラマに読み込むためのHEXファイルに

変換するための**OH86**がある．その他，8080/8085用プログラムを8086用に変換する**CONV-86**も用意されている．

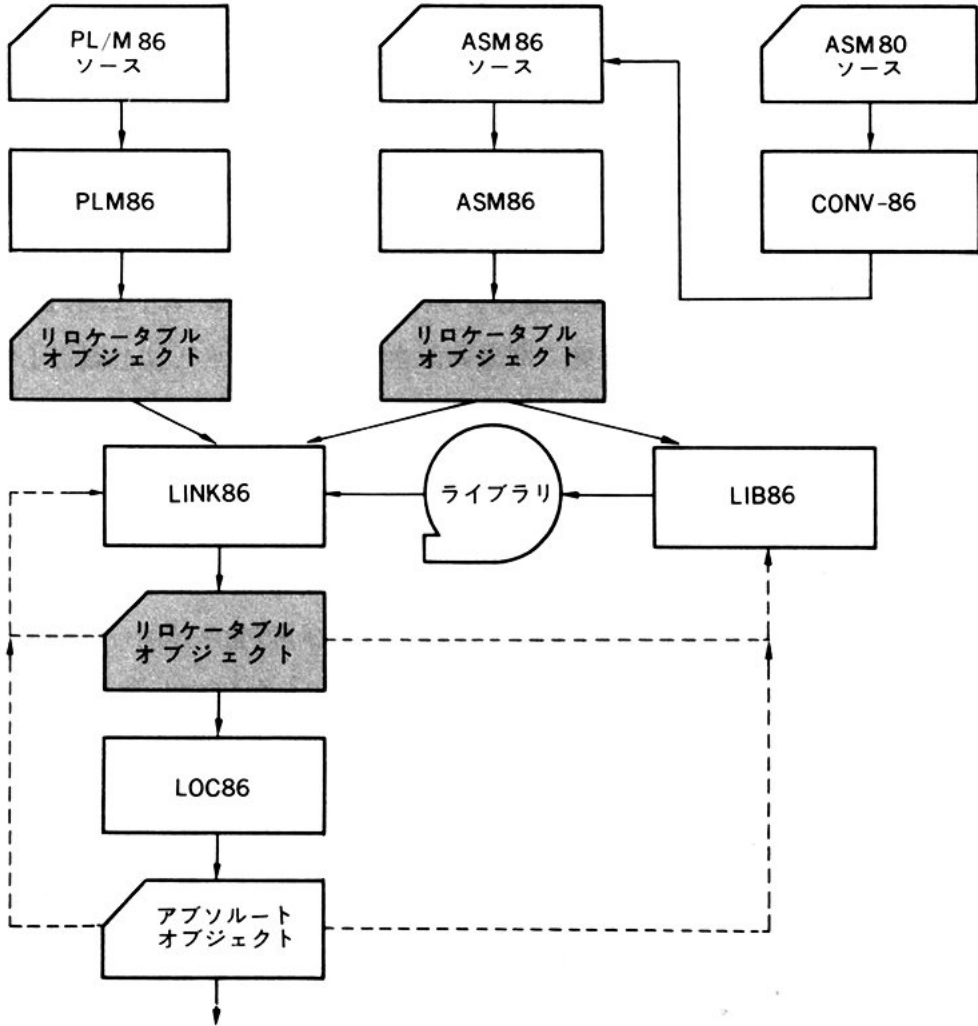



図 10・1 プログラムの開発フロー

10・2 インサーキットエミュレータ（ICE86）

インサーキットエミュレータが登場するまでは，マイクロコンピュータシステムはハードウェアとソフトウェアの開発を別々に行い，両方が完成した時点でそれらを組み合わせて，相互にデバックすることが行われていた．インサーキットエミュレータはユーザシステムのハード/ソフトがともに不完全な段階から，ユーザシステムで不足しているメモリやI/O等をMDS開発装置から借用して実行できるようになっており，次のような数々の強力なデバック機能を提供する．

（1） ICE86ケーブルの先端の40ピンソケットがユーザシステムの8086CPUと置き換えられ，そのピンから見て開発装置全体が大量のメモリと周辺I/O装置を持ったCPUとして働く．

（2） メモリマッピングの機能があり，ユーザシステムを動作させるためのメモリとして，ユーザシステム上のメモリか開発装置内のメモリかを自由に選択でき，1Kバイト単位でコマンドにより前もってそのマッピングを設定できる．

（3） エミュレーションコマンドとしてGOおよびSTEPがあり，GOは指定した開始アドレスから，二つまでのブレークポイントを設定して実行できる．STEPはシングルステップでの実行を行う．

（4） コマンドでのアドレスの指定はラベルで行うことができ，フルシンボリックデバッキングが可能である．

（5） 二つまでのトレースポイントが設定でき，トレース情報の収集のスタート/ストップ条件を設定し，約300バスサイクルまでの情報をトレースメモリに格納でき，後でその履歴を調べることができる．

また，マクロ機能があり，頻繁に使用するコマンドシーケンスをあらかじめ定義しておくことにより，そのマクロ名と最大10個までのパラメータを渡すことにより簡単に指定できる．複合コマンドはコマンドの条件付き実行（IF）で，ある条件に出合うまでか，あるいはそれらが指定回数実行されるまでそのコマンドを実行する．トレースメモリおよびメモリ内のオブジェクトコードを調べる場合，ディスアセンブルが可能で，アセンブラのニーモニックでのチェックができる．

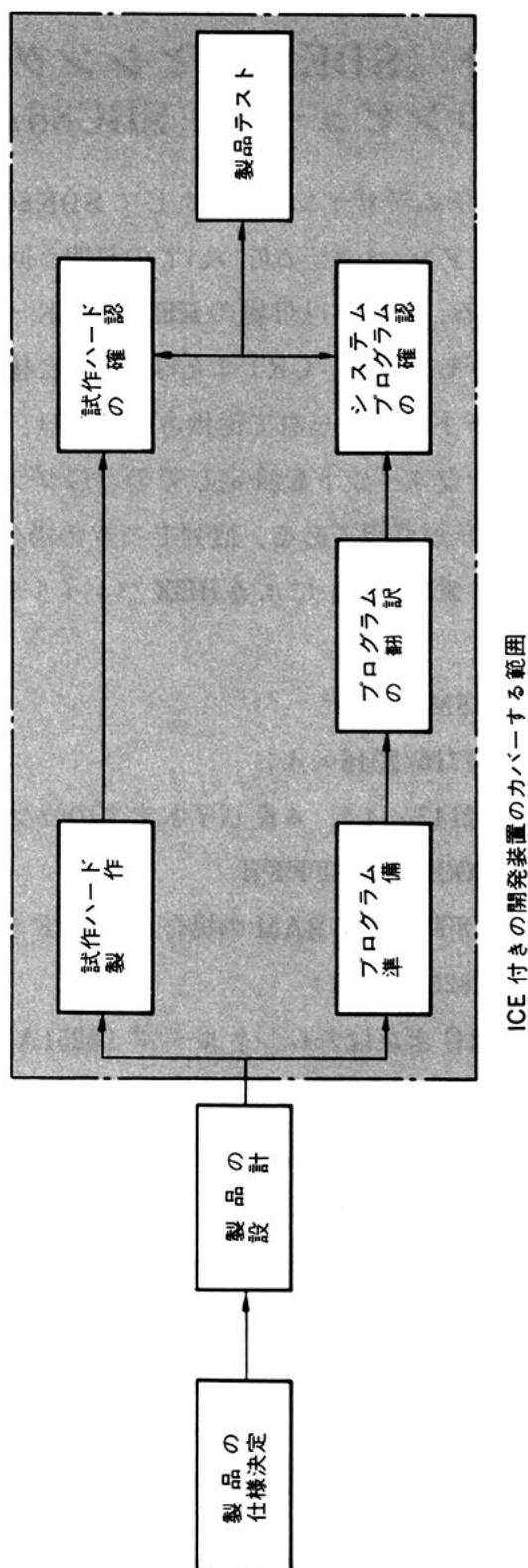



図 10・2　ICE 付き開発装置による製品の開発サイクル

10・3 評価用キット（SDK86）とシングルボードコンピュータ（SBC86/12A）

〔1〕 **SDK86**　8086用のシステムデザインキットとしてSDK86が発売されており，組立て要領に関するマニュアル，システムについての説明と回路図の載ったユーザーズガイドが付属している．モニタは付属のHEXキーボードおよび8桁のLED表示のためのキーパッドモニタと，CRTまたはTTYと接続して使用する直列モニタがそれぞれ2Kワードに納められて提供されており，メモリの表示/変更，ブロックMOVE，ブレークポイントを設定してのプログラムの実行およびシングルステップによる実行等が可能である．直列モニタの場合はTTYと接続することにより，紙テープリーダパンチャによるHEXファイルの入/出力が可能である．以下に仕様例を示す．

・CPU：8086　5 MHzまたは2.5MHz選択

・メモリ：ROM　8Kバイト（2716/2316×4）
　　　　　RAM　2Kバイト（2142×4）　4Kバイトまでのソケット

・アドレッシング：ROM　FE000H～FFFFFH
　　　　　　　　　RAM　0～7FF（追加RAMの場合0～FFFまで）

・入/出力：並列入出力　48本（8255A×2）
　　　　　直列入出力　RS232Cまたはカレントループ（8251A）

・割込み：256ベクタ割込み．マスカブル/ノンマスカブル/トラップ

〔2〕 **SBC86/12A**　86/12Aは8086をマキシマムモードで使用し，バスコントローラ（8288）およびバスアービタ（8289）を使用した完全マルチバスコンパチブルなボードである．32KバイトのデュアルポートRAM* を実装しており，オンボードのCPUからだけでなくマルチバスを通して他のCPUからも，この同一メモリのアクセスが可能になっている．以下に仕様を示す．

・CPU：8086　5 MHz　1.2μs命令サイクル（キューから400ns）

・メモリ：ROM　16Kバイト（拡張モジュールにより32Kまで可能）

---

* ボード上のCPUからだけでなく，マルチバスを通して外部CPUからもアクセスできるように構成したRAM．

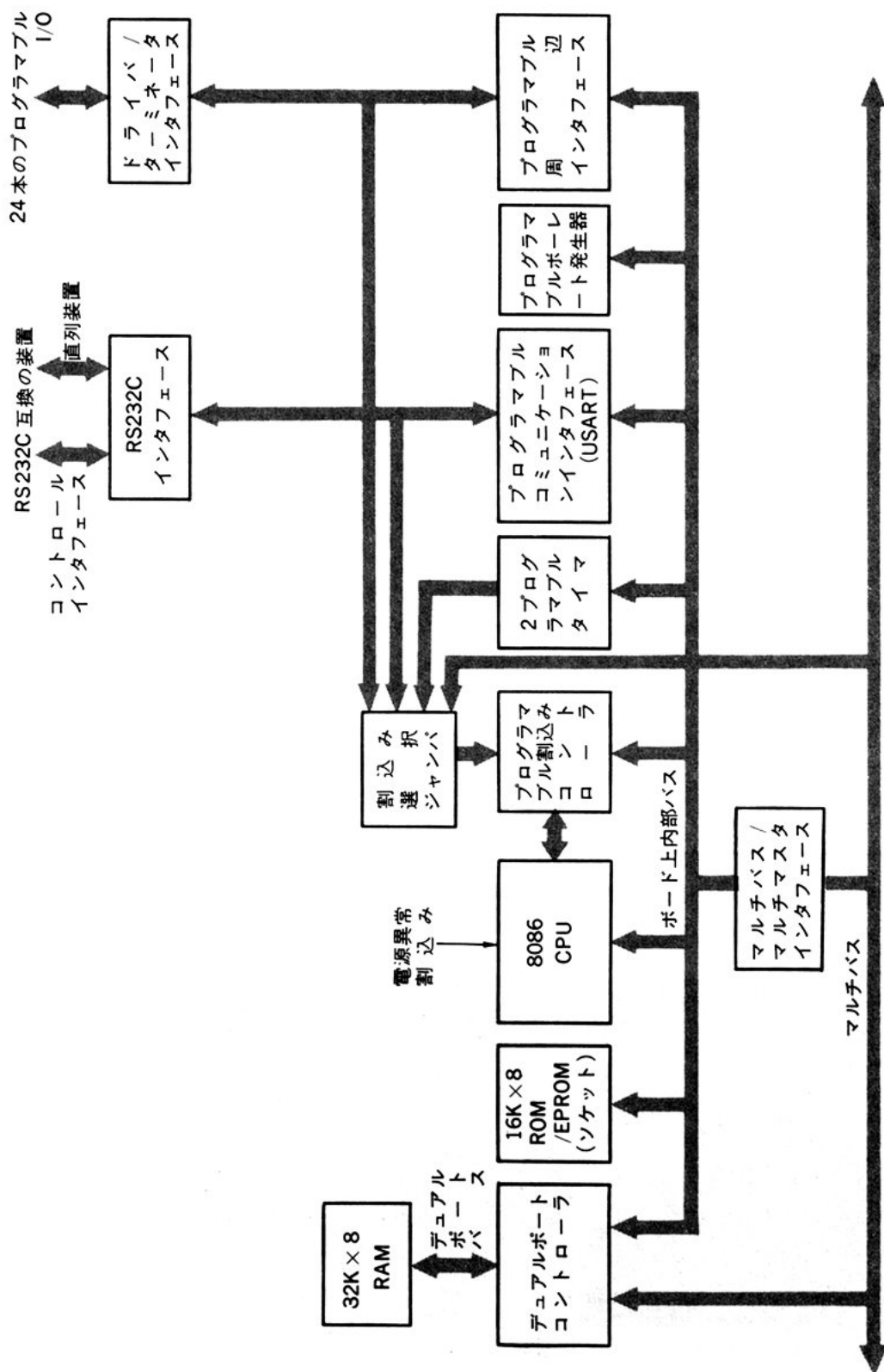



図 10・3　SBC86/12Aのブロックダイヤグラム

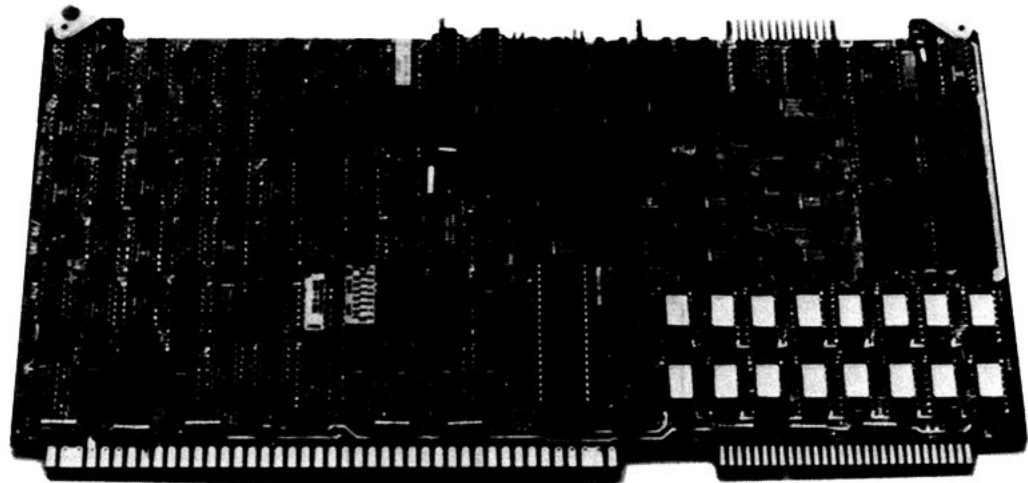
RAM　32K バイト（拡張モジュールにより 64K まで可能）

I/O：直列インタフェース　RS 232C（8251A）

並列インタフェース　24本入出力ポート（8255A）

タイマ　16 ビット 2 個（8253-1 個はボーレートジェネレータとして使用ずみ）

・割込み　9 レベル（8259A 実装）

マルチバスボード SBC86/12A の写真

11. プログラミング言語/リアルタイムモニタ

8086の開発装置MDS上で使用可能なプログラミング言語のうち，アセンブラ(ASM86)とコンパイラ(PLM86)を紹介している．また，86用のリアルタイムモニタ(RMX86)の概要についても記述している．近々PASCAL86も入手可能になる予定である．

11・1 アセンブラ（ASM86）

8086/8088のアセンブラは，従来高級言語でのみ使用可能であったデータのストラクチャ機能やプロセデュアの使用など一歩高級言語に近づいたアセンブラといえる．また，たとえばMOV命令のように一つの命令で多数のバリエーションがある場合に，プログラムは単にMOVとソースおよびディスティネーションオペランドを指定するだけで，あとはアセンブラがそのオペランドの属性に合った最適な機械語を発生してくれる．

〔1〕 **ステートメント**　ステートメントのフォームは，

ラベル：（プリフィックス） ニーモニック （オペランド）（：コメント）

の形式をとりラベルは31文字まで．読みやすさを改善するため任意にブランクや下線を入れることができる．ここでカッコ付きは指定がオプションであることを示す．プリフィックスとは，セグメントオーバライド，バスロックおよびリピート等のプリフィックスで，この命令の後の動作を規定するものである（6章参照）．

〔2〕 **データの定義**　三つのアセンブラ指令DB（バイト），DW（ワード），DD（2ワード定義）により確保する領域の数やその初期値等を規定する．

〔3〕 **レコード**　ASM86は，バイトまたはワード内の個々のビットまたはストリングをシンボリックに定義可能である．図11・2に示すように，1バイトのEMP_BYTEをさらにYRS，SEXおよびSTATUSに分けて定義し，TESTのところでその部分だけをマスクして取り出して調べるといった使い方ができる．

〔4〕 **ストラクチャ**　ストラクチャはデータフィールドの集まりに対し名前や属性（長さ，タイプ等）を与えるもので，DB，DWおよびDDを使って定義される．ストラクチャには領域は割り当てられず，あるフィールド名がベースアドレスとともに命令中で参照されたときに，メモリの特定のエリアと一緒になる．図11・3に示すように，MASTERおよびTXNが命令中で参照されたときにストラクチャ中のRATEがそれと一緒になる．

〔5〕 **プロセデュア**　同じルーチンがプログラム中で反復使用される場合にプロセデュアとして定義し，プログラムの他の部分からCALLにより呼び出す．

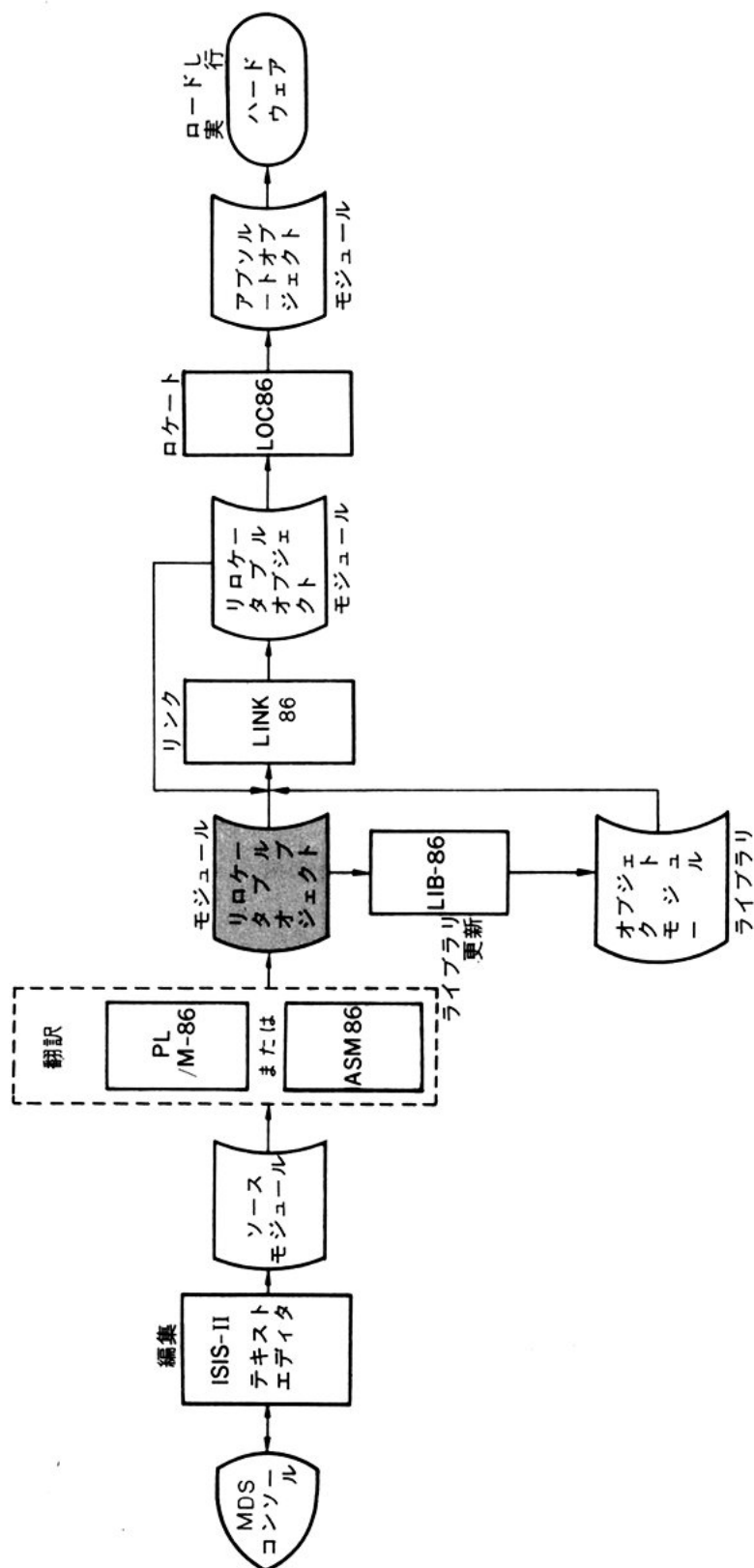



図 11・1 ソフトウェアの開発過程

```
        EMP_ BYTE DB  ?                  ;1BYTE, UNINITIALIZED
        ;BIT DEFINITIONS:
        ;      7-2   :YEARS EMPLOYED
                1    :SEX(1=FEMALE)
                0    :STATUS(1=EXEMPT)
        EMP_BITSRECORD                   ;RECORD DEFINED HERE
        &          YRS_EMP:6,
        &           SEX:1,
        &           STATUS:1
        .
        .
        .
        ;SELECT NONEXEMPT FEMALES EMPLOYED 10+YEARS

        MOV         AL, EMP_BYTE         ;KEEP ORIGINAL INTACT
        TEST        AL, MASK SEX         ;FEMALE?
        JZ          REJECT               ;NO, QUITE
        TEST        AL, MASK STATUS      ;NONEXEMPT?
        JNZ         REJECT               ;NO, QUIT
        SHR         AL, CL               ;ISOLATE YEARS
        CMP         AL, 11               ;>=10YEARS?
        JL          REJECT               ;NO, QUIT
        ;PROCESS SELECTED EMPLOYEE
        .
        .
        .
REJECT: ;PROCESS REJECTED EMPLOYEE
        .
        .
        .                                ;RECORD USED HERE
        MOV         CL, YRS_EMP          ;GET SHIFT COUNT
```

図 11・2 ASM86 レコードの例

```
EMPLOYEE       STRUC
  SSN          DB 9   DUP(?)
  RATE         DB 1   DUP(?)
  DEPT         DW 1   DUP(?)
  YR_HIRED     DB 1   DUP(?)
  EMPLOYEE     ENDS

MASTER         DB 12  DUP(?)
TXN            DB 12  DUP(?)

;CHANGE RATE IN MASTER TO VALUE IN TXN.
               MOV    AL, TXN. RATE
               MOV    MASTER. RATE, AL

;ASSUME BX POINTS TO AN AREA CONTAINING
;    DATA IN THE SAME FORMAT AS THE EMPLOYEE
;    STRUCTURE. ZERO THE SECOND DIGIT
;    OF SSN
               MOV    SI, 1  ;INDEX VALUE OF 2ND DIGIT
               MOV    [BX]. SSN[SI], 0
```

図 11・3 ASM86 ストラクチャの例

11・2 PL/M-86コンパイラ

PL/M-86は8086/8088用の高級言語で，8080/8085用のPL/M-80とソースレベルで上位コンパチブルになっている．高級言語を使用することにより，プログラムの開発時間を短縮し，ソフトウェアのメンテナンスコストを軽減†している．記述は簡単な英文に近い文と算述表現式により行う．

〔1〕 **プログラムの構成**　PL/M-86のプログラムは，データの定義や実行を指示するステートメントとコメントの連続として記述される．各ステートメントはセミコロン(;)で終了し，コメントは“/*”と“*/”の間に囲まれて記述され，プログラム中の任意の場所に挿入可能である．

〔2〕 **データ定義**　プログラムは通常データの定義で始まり，そのおのおのの要素はスカラーと呼ばれ，31キャラクタまでの名前を持つことができ，五つのタイプ，すなわちバイト，ワード，インテジャ，リアルおよびポインタが使用できる．表11・1にそのデータタイプの一覧表を示す．変数定義は宣言文で行う．

DECLARE　スカラー名　タイプ;

また，配列の定義は次のようにして行い，たとえばその6番目の要素はDATA(5)のように表す．

DECLARE　DATA (12) REAL;

関連したデータ要素の集まりを定義するものとしてSTRUCTUREがあり，その各要素はDATA.LENGHTのようにドットを付けて表す．

```
DECLARE  DATA  STRUCTURE
  (LENGTH WORD,WIDTH BYTE,HEIGT WORD);
```

〔3〕 **割当て文と各種演算子**　割当て文は下記の形式をとり，表現式を評価した後，その結果を左側の変数に入れる．表現式は定数のほか算述演算，関係演算および論理演算式が可能で，表11・2にその一覧表を示す．

変数名＝表現式

〔4〕 **プログラムの流れに関するステートメント**　プログラムの流れをコン

† プログラム記述の長さが短くなり，読みやすくメンテナンスが簡単なため．

トロールする文としてIF文とDO文がある．IF文は記述中の関係表現式が満足される場合にステートメント1を，そうでない場合にステートメント2を実行し，条件分岐の働きをする．

IF 関係表現式 THEN ステートメント1; ELSE ステートメント2;

DO文は，ある条件が満足されている間，あるルーチンを繰返し実行するもので，単純DO，DO CASE，DO WHILEの三つがあり，DO CASEはDO文中に記述されている実行文のうち，CASEの条件に合うものだけが選択的に実行される．DO WHILEはWHILEの後の表現式が“真”の間そのルーチンを繰返し実行する．

また，プログラムのコントロールを移すステートメントとしてGOTOとCALLがある．

GOTO START

CALL プロセデュア名*（パラメータリスト）

プロセデュアはプログラムの始めに定義されるもので，パラメータリストはそのプロセデュアに渡される変数である．

〔5〕 **プロセデュア**　プロセデュアは，複雑なプログラムを一つの機能を持ったいくつかのサブプログラムに分割し，メインルーチンからCALL文によって呼び出して実行する．使用法については図11·4の例を参照のこと．

* プログラムの他の部分でプロセデュアとして定義されているサブルーチン名.

表 11・1 PL/M-86 のデータのタイプ

タイプ	バイト数	データ範囲	使用
BYTE	1	0～255	符号なし整数, キャラクタ
WORD	2	0～65,535	符号なし整数
INTEGER	2	−32,768～ +32,767	符号付き整数
REAL	4	1×10^{-38}～ $3.37\times10^{+38}$	浮動小数点数
POINTER	2/4	N/A	アドレス操作

表 11・2 PL/M-86 の表現式

表現式	オペレータ	結果
算述演算子	+, −, *, /, MOD	数値
関係演算子	>, <, =, >=, <=	"真"-FFH "偽"-OH
論理演算子	AND, OR, XOR, NOT	8/16-ビットストリング

```
/*DECLARATION OF A TYPED PROCEDURE THAT
  ACCEPTS TWO REAL PARAMETERS AND RETURNS A REAL VALUE*/
  AVG:PROCEDURE(X, Y)REAL;
      DECLARE(X, Y)REAL,
      RETURN(X+Y)/2.0;
      END AVG;

/*ACTIVATING A TYPED PROCEDURE*/
LOW=2.0;
HIGH=3.0;
TOTAL=TOTAL+AVG(LOW, HIGH); /*2.5IS ADDED TO TOTAL*/

/*DECLARATION OF AN UNTYPED PROCEDURE
  THAT ACCEPTS ONE PARAMETER*/
TEST:PROCEDURE(X);
  DECLARE X BYTE;
  IF X=OH THEN
      COUNT=COUNT+1;
  END TEST;

/*ACTIVATING AN UNTYPED PROCEDURE*/
CALL TEST(ALPHA); /*COUNT IS INCREMENTED
                    IF ALPHA=0*/
```

図 11・4 PL/M-86 プロセデュアの例

11・3　リアルタイムモニタ（RMX86）

リアルタイムモニタは，プロセスコントロールのような実時間でランダムに発生する事象をモニタし，多重におこってくる処理事項を検知し，記憶し，それらをどのような優先準位で処理するかなどをコントロールするプログラムである．たとえば，コンピュータシステムが相対的に重要度の低い仕事を処理している最中に，緊急に処理しなければならない仕事が発生した場合，現在処理中の仕事を一時中断し，その重要度の高い仕事を優先して処理し，その終了後に中断していた前の仕事に戻り，処理を続行する．また，RMX86はこれらのリアルタイム処理に加えて，通常の汎用OSの持っているファイル管理機能等も兼ね備えている．

RMX86の構成は図11・5に示すように，そのもとになる核の部分に，マルチタスキング，マルチプログラミング，タスク内交信，割込み処理，およびエラーチェック等の基本的な機能が入っており，システムに常駐させるので，2Kバイト程度とできるだけ短いプログラムになっている．基本I/Oシステム（BIOS）は，I/O装置の種類やファイルフォーマットに依存しないデータ操作の機能を，拡張I/Oシステムは同期I/Oコールや高度なジョブ管理などの機能を提供する．フロッピーやハードディスクのような大容量記憶装置からプログラムコードやデータをRAMにロードするためのアプリケーションローダや，CRTターミナル等と人間との間のインタフェースを行うマンマシンインタフェース等のプログラムがオプションとして必要に応じて付加される．また，ユーザの書いたアプリケーションプログラムもこれらのプログラムと必要に応じて結合される．

図11・7にRMX86オペレーティングシステムの中からアプリケーションソフトが必要とする部分だけを取り出して結合し，アプリケーションシステム全体のプログラムを構成する例を示す．この中で核以外の部分はすべてオプションで，そのアプリケーションで不要の場合は結合する必要はない．たとえばデバッカの場合はシステムの初期段階および，新しい機能が追加されたときなどのデバック用に使用され，開発が完了した時点では取り除くことができ，アプリケーションプログラムのサイズを削減できる．また，タスク間のコマンドやデー

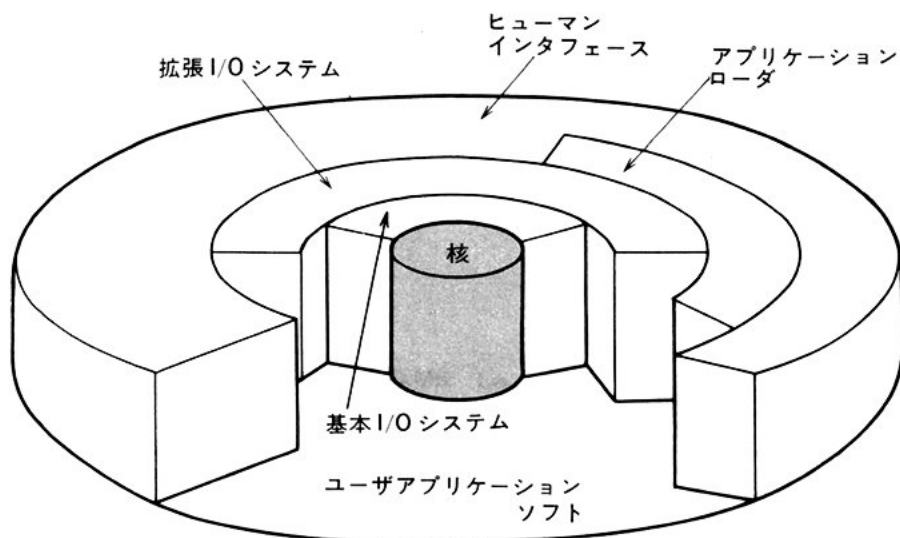



図 11・5 リアルタイムモニタ RMX86 の構成

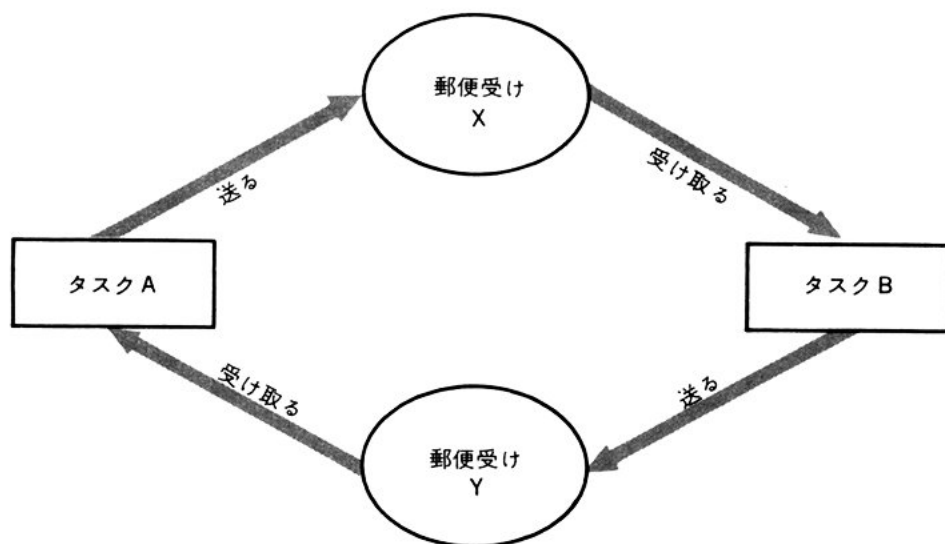



図 11・6 タスクA・タスクB間のメッセージ交換

タの交換は，図11·6のように，一方のタスクが郵便受けにデータを送り，他方のタスクがそれを受け取るという方式で行われる．

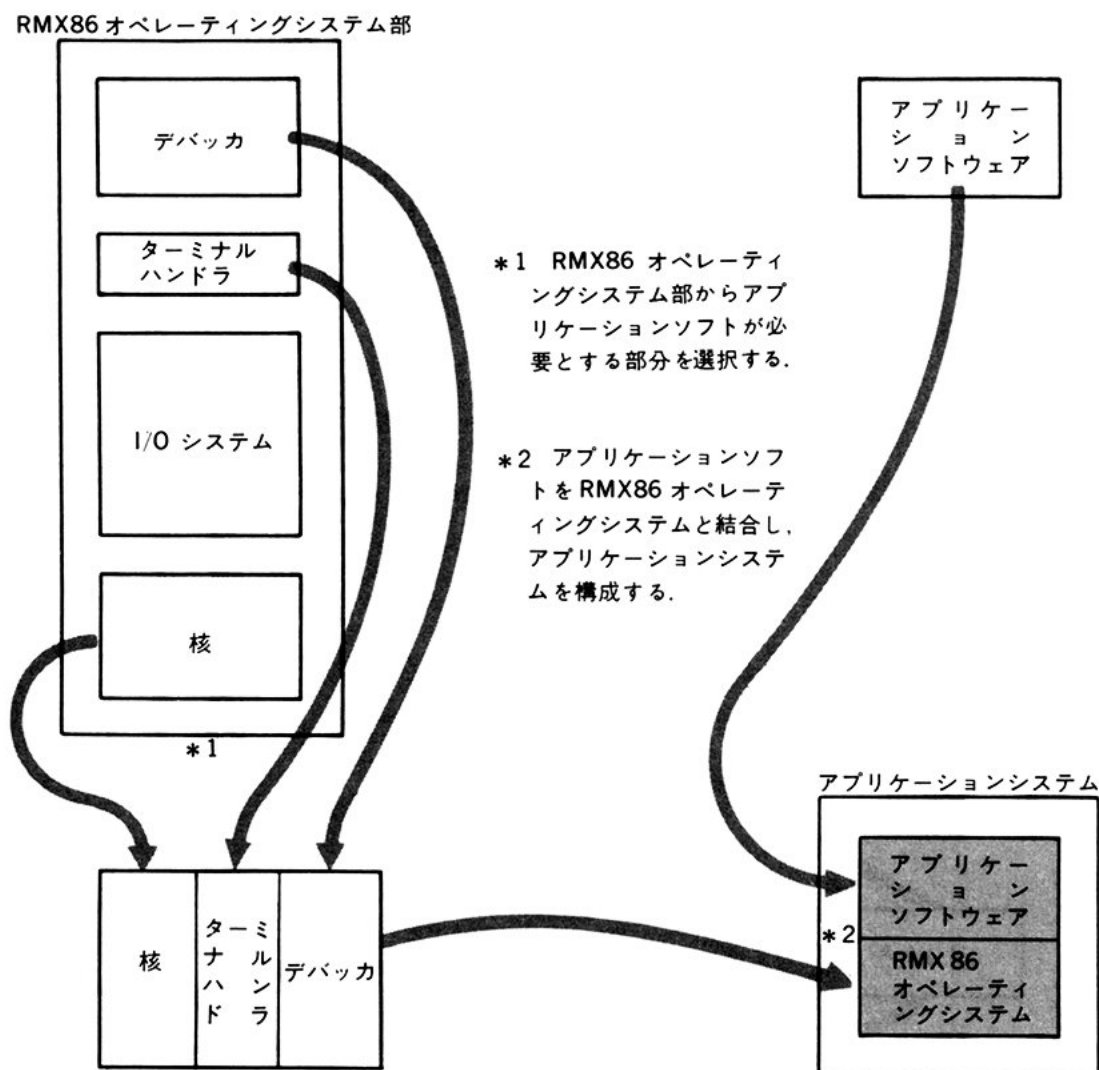



図 11・7　リアルタイムモニタ（RMX86）の構成

12. ファミリプロセッサによる機能の拡張と8086の発展方向

8086を中心とし，8087（高速演算プロセッサ）および8089（I/Oプロセッサ）と組み合わせて，より高度なマルチプロセッサシステムを構成可能である．これらのものは86の演算およびI/O 処理機能を飛躍的に向上させる．また，86を基本とした16ビットCPUの将来の発展方向，32ビットCPUについても簡単に述べる．

12・1 高速演算プロセッサ（コ・プロセッサ）8087

8087数値データプロセッサは8086/8088の数値演算能力を補うもので，図12・1に示すように結線することにより8086と一体になって動作する．表12・1は8087によって付加される演算機能とその処理速度を8086のソフトウェアでエミュレートした場合との比較で示している．

8087はCPUのキュー状態情報をもらい，CPUに同期して自分に関係のある命令を受け取り，デコードする．すなわち，すべての8087の機械語命令の最初の5ビットはESCクラス（D8H～DFH）の命令になっており，コントロールユニットはそれ以外の命令はすべて無視する．8086のプログラムが87の演算機能を使用する場合の手順は，図8・6に示したように，CPUがエスケープコードを含んだ命令をフェッチするときに，それと並列にコ・プロセッサもそのコードを同時にフェッチし，CPUの助けを借りてその命令のデコードおよび実行を行う．その後はCPUは別の仕事を並行して実行することができ，その演算結果が必要になるところにWAIT命令を挿入することにより，$\overline{\text{TEST}}$信号との組合せによりCPUと同期をとってその結果の受渡しを行うことができる．8087がエラー等の例外状態を検出した場合はプログラマブル割込みコントローラ**8259A**を通してCPUに割込みをかけ，その例外処理を行う．

また，8087がデータ転送のためにローカルバスのコントロール権を得るには，ホストCPUの$\overline{\text{RQ}}/\overline{\text{GT}}$を使用する．

8087が扱うデータのタイプとその数値の範囲を表12・2に，またそのデータのフォーマットを図12・3に示す．固定小数点の2進数で64ビットまで，10進数で18桁までの数値の扱いが可能である．

8087は，IEEEが答申中の工業界標準としてのミニコン/マイコンのための浮動小数点演算のフォーマットに準拠しており，他のコンピュータとの間での数値演算プログラムのポータビリティ（可搬性）を促進するものである．

8087のアプリケーションは，18桁までの10進演算，浮動小数点演算，三角関数および指数関数などの強力な演算能力のため，広くビジネスデータ処理，プロ

セスコントロール，数値制御，ロボット，ナビゲーション，グラフィックターミナルおよびデータ集録などの応用が考えられる．

図 12・1　8087 の使用

表 12・1　8087 と 8086 のソフトとの演算速度の比較

命令の種類	概略の実行時間〔μs〕(5 MHz クロック)	
	8087	8086 によるソフトウェアエミュレーション
乗　算　(単精度)	19	1 600
乗　算　(倍精度)	27	2 100
加　算	17	1 600
除　算　(単精度)	39	3 200
比　較	9	1 300
ロード　(単精度)	9	1 700
ストア　(単精度)	18	1 200
開　平	36	19 600
タンジェント	90	13 000
指数演算	100	17 100

コントロールユニット
数値実行ユニット
コントロールワード
ステイタスワード
NEU 命令
データバス
データバッファ
オペランドキュー
マイクロコードコントロールユニット
指数バス
指数モジュール
フラクションバス
プログラマブルシフタ
インタフェース
16
アリスメティックモジュール
68
16
16
64
一時的レジスタ
ステイタス
アドレスバス
アドレッシングとバストラッキング
例外ポインタ
タグワード
レジスタスタック
(7)
(6)
(5)
(4)
(3)
(2)
(1)
(0)
80 ビット

図 12・2　8087 のブロックダイヤグラムとピン構成図

表 12・2 8087のデータタイプとその範囲

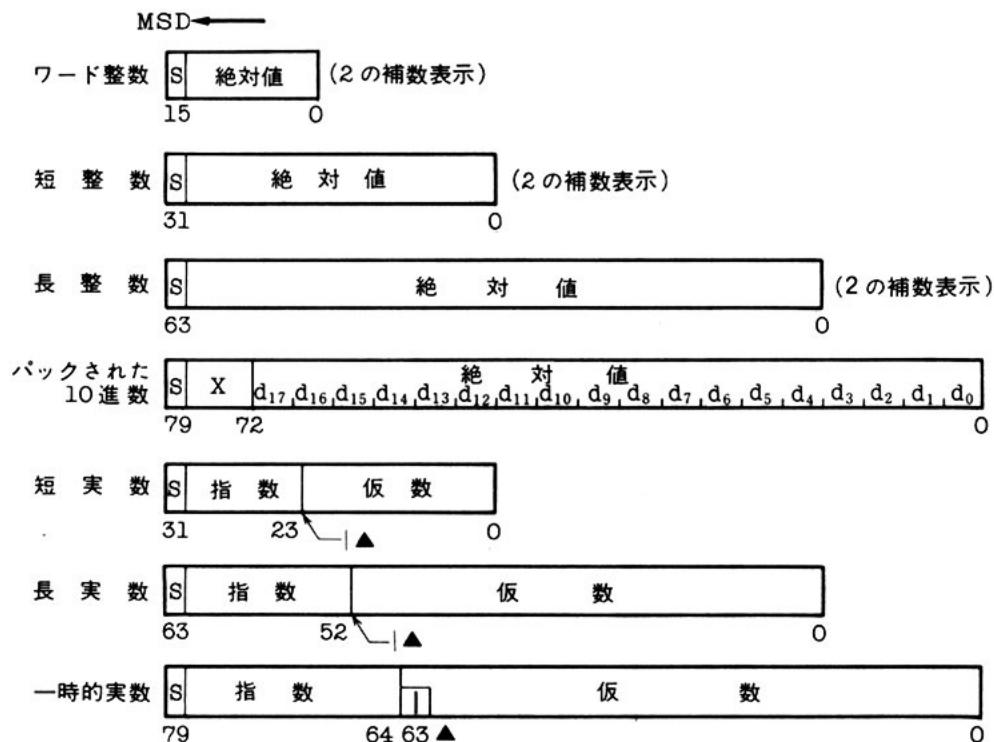
データタイプ	ビット数	有効桁数（10進）	概略範囲（10進）
ワード整数	16	4	$-32\,768 \leqq X \leqq +32\,767$
短整数	32	9	$-2\times10^{9} \leqq X \leqq +2\times10^{9}$
長整数	64	18	$-9\times10^{18} \leqq X \leqq +9\times10^{18}$
パックされた10進数	80	18	$-99\cdots99 \leqq X \leqq +99\cdots99$(18ディジット)
短実数	32	6～7	$8.43\times10^{-37} \leqq \lvert X\rvert \leqq 3.37\times10^{38}$
長実数	64	15～16	$4.19\times10^{-307} \leqq \lvert X\rvert \leqq 1.67\times10^{308}$
一時記憶の実数値	80	19	$3.4\times10^{-4932} \leqq \lvert X\rvert \leqq 1.2\times10^{4932}$

〔注〕 1. S：符号（0＝正, 1＝負）
2. d_n：10進数（1バイト当り2ディジット）
3. X：何も意味を持たない
4. ▲：暗黙の2進の小数点
5. |：一時的実数の場合に使用される指数部の整数ビット

図 12・3 8087のデータ形式

表 12・3　8087 演算命令一覧表

加　　算	
FADD	実数加算
FADDP	実数加算とポップ
FIADD	整数加算
減　　算	
FSUB	実数減算
FSUBP	実数減算とポップ
FISUB	整数減算
FSUBR	実数逆減算
FSUBRP	実数逆減算とポップ
FISUBR	整数逆減算
乗　　算	
FMUL	実数乗算
FMULP	実数乗算とポップ
FIMUL	整数乗算
除　　算	
FDIV	実数除算
FDIVP	実数除算とポップ
FIDIV	整数除算
FDIVR	実数逆除算
FDIVRP	実数逆除算とポップ
FIDIVR	整数逆除算
その他の命令	
FSQRT	開　平
FSCALE	スケール
FPREM	部分剰余
FRNDINT	整数へ丸め込み
FXTRACT	指数および仮数の抽出
FABS	絶対値
FCHS	符号反転

12・2 高速I/Oプロセッサ8089

マイクロコンピュータシステムが高度なものになるにつれて，従来のように I/O のコントロールまで含めすべての仕事を一つのCPUにさせようとすると，CPUの時間のほとんどがこれらのI/O動作に専有されることになり，そのシステム全体のスループットを低下させる原因になる．そして，最近ではこれらのシステムに接続される周辺装置も高速のデータ転送を必要とするようになってきており，特に実時間動作の場合には，要求に対する即時のサービスを必要とし，メインCPUとの並列処理が要求される．

8089 I/O プロセッサは8086 CPUと組み合わせて，このような問題点を解決するために開発されたもので，メインフレームで使用されているインテリジェントI/O サブシステムやチャネルコントローラ等の思想を8086 CPUを中心としたマイクロコンピュータシステムの領域に適用したものである．

図8・5に示したように，8089を使用したシステムではCPUから周辺I/Oを分離し，必要なときだけCPUが介在するようにすることによりシステムのスループットは大幅に向上する．表12・4にデータ転送レートを示す．

8089を使用した場合のバスの構成は図12・4に示すように，二つの独立したI/Oチャネルと，それぞれに専用のレジスタセットおよび命令ポインタを持っており，独立してDMA転送や一連の命令を実行する．8089のレジスタは図12・5に示すように全く同じ2組のレジスタ群を持ち，GA，GBはともに20ビットで，システムバスまたはローカルバスのいずれかを指定することができる．そしてこれらはDMA転送時にはソースおよびディスティネーションアドレスの指示に使用され，自動的にインクリメントされる．また，GCレジスタはコード変換（ASCII ⇄ EBCDIC等）などの翻訳動作に使用され，DMA転送時のテーブル参照のポインタとして働く．

その他，転送中のデータのビットごとの操作，テストやマスクをかけることも可能である．CPUとIOPの間のコミュニケーションはチャネルアテンション（CA）と割込みラインにより扱われ，タスクプログラム，ステイタス情報および

パラメータ等は割り当てられたメモリブロックを介して相互に伝達される.

このようにして，DMA動作は通常のメモリ・I/O間だけでなく，メモリ・メモリ間，およびI/O・I/O間にも拡張され，高速のブロック転送などが可能である.

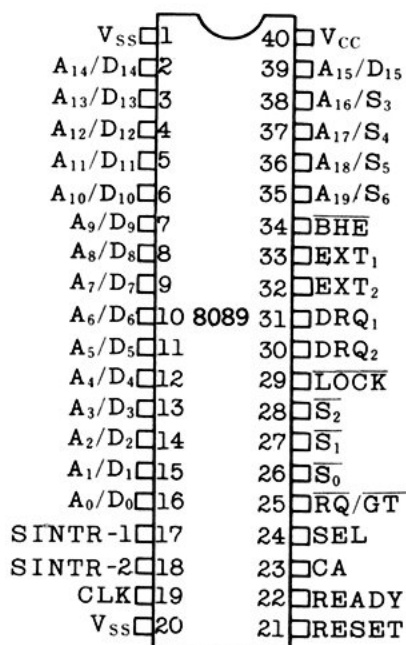
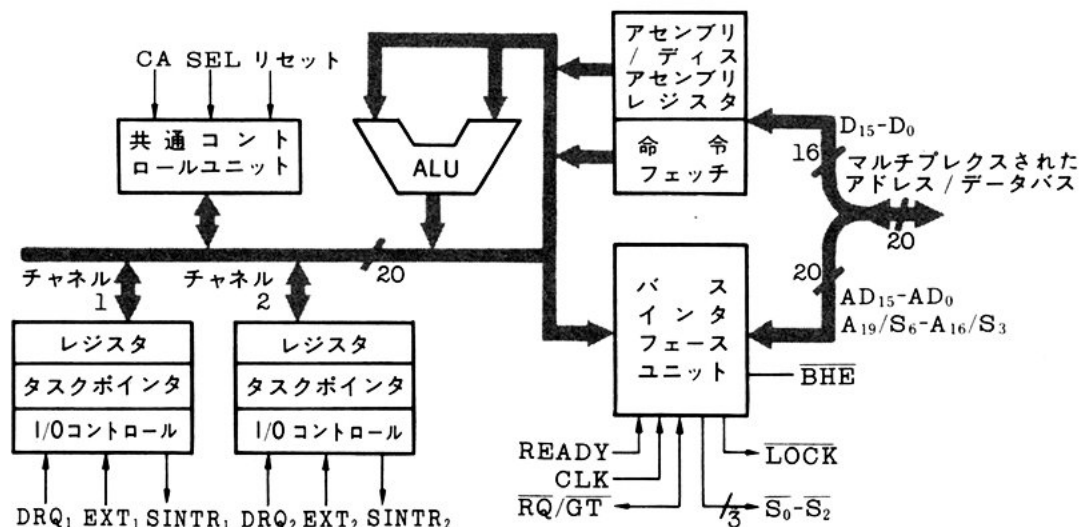



図 12・4 808910プロセッサのブロックダイヤグラムとピン構成図

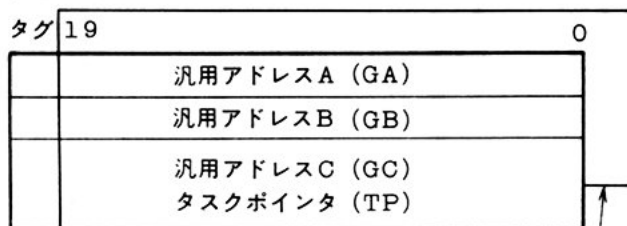
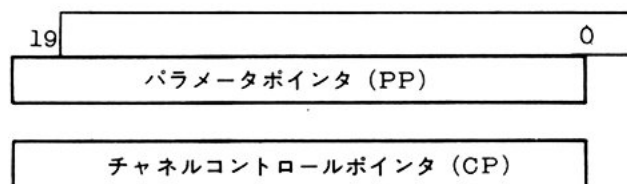
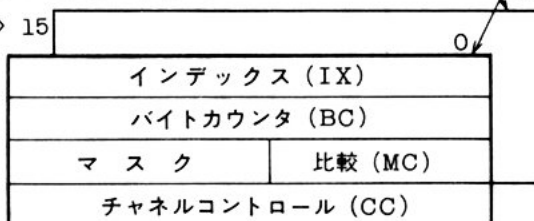



図 12・5　8089 レジスタの構成

表 12・4　8089 のデータ転送レート

	ローカルバス		リモート	
	バイト	ワード	バイト	ワード
バンド幅〔K バイト/s〕	830	1 250	830	1 250
レイテンシイ〔μs〕	1.0/2.4*	1.0/2.4*	1.0/2.4*	1.0/2.4*
システムバス使用時間〔μs〕	1転送当り 2.4	1転送当り 1.6	1転送当り 0.8	1転送当り 0.8

* チャネルが要求の待ち状態にあるとき 1.0 μs，その他の場合および他のチャネルとインターリーブする場合 2.4 μs

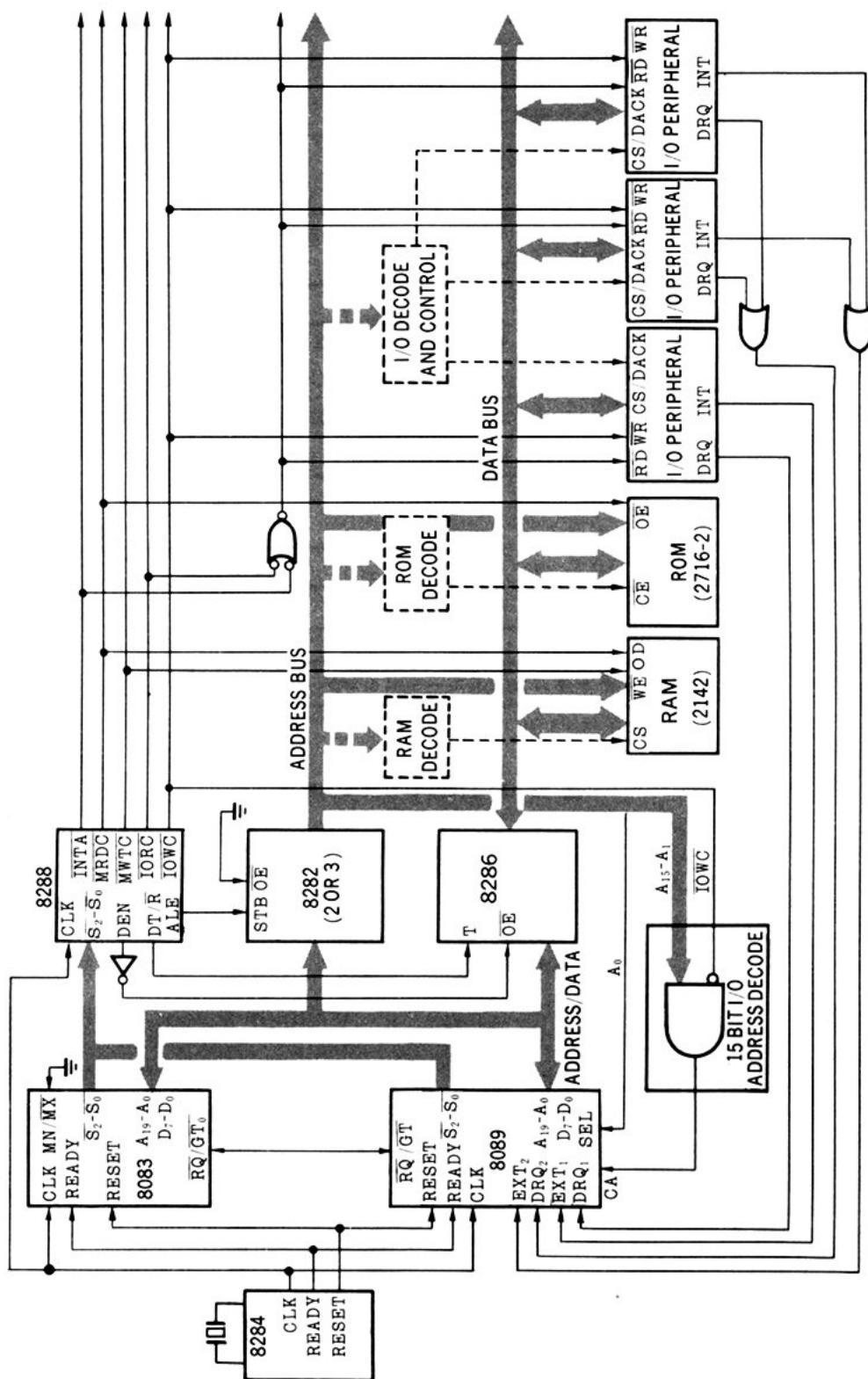



図12・6 8088/8089のローカルモード構成例

12・3　8086の将来の発展方向

8086を中心とし，高速演算チップ（8087）およびI/Oプロセッサ（8089）を加えたシステムは従来の8ビットシステムと比較して処理能力はほぼ1桁向上したといえるが，むしろこれは16ビット以上のマイクロコンピュータの始まりと見るべきである．次のステップとしては，システムの高度化に伴うソフトウェア開発コスト増大の軽減のため，図12・7に示すようなステップでソフトウェアをシリコン上に集積するための開発が行われている．

まず，第1ステップとしては，オペレーティングシステム（OS）の核の部分を，次のステップとして高級言語をファームウェアとしてチップ上に組み込む計画が進行している．ここでは8086の後に続いて発表されている3種類のCPUについて簡単に紹介する．表12・5はこれらのCPUの概要の一覧表である．

〔1〕 **iAPX 186/188**（マイクロミディ）（16ビットCPU）　iAPX 186/188は，8086/88システムの場合の発振器等の外付け回路と，OSの核の部分をCPUとともに1チップに集積したもので，システム価格の低減と上位互換性を狙ったものである．

〔2〕 **iAPX 286**（マイクロマキシ）（16/32ビットCPU）　iAPX 286は8086とプログラムコードの互換性を有し，次のような点が強化されている．iAPX 286のブロックダイヤグラムを図12・8に示す．

- 物理アドレス空間が16Mバイトに拡張された．
- タスク当り，1ギガバイトまでの仮想メモリスペース管理機能．
- パイプライン構造により，標準8086の5倍の処理速度を有する．
- チップ上に組み込まれたメモリ管理と保護機構および多レベルのソフトウェア保護機能．
- 86の拡張された命令セットと，チップ上にOSの核の部分を組み込んでいる．

〔3〕 **iAPX 432**（マイクロメインフレーム）（32ビットCPU）iAPX 432は32ビットのCPUで，マイクロシステムの領域にメインフレームの機能を導入したものである．図12・9に示すように8086ファミリやマルチバスとの間のインタフェー

スのためのインタフェースプロセッサ，データ処理の機能的分散に使用されるデータプロセッサ，および記憶モジュールの三つのチップで構成される．また高級言語やオペレーティングシステムの機能をシリコン上に集積することにより，莫大なソフトウェア開発コストを軽減するよう設計されている．

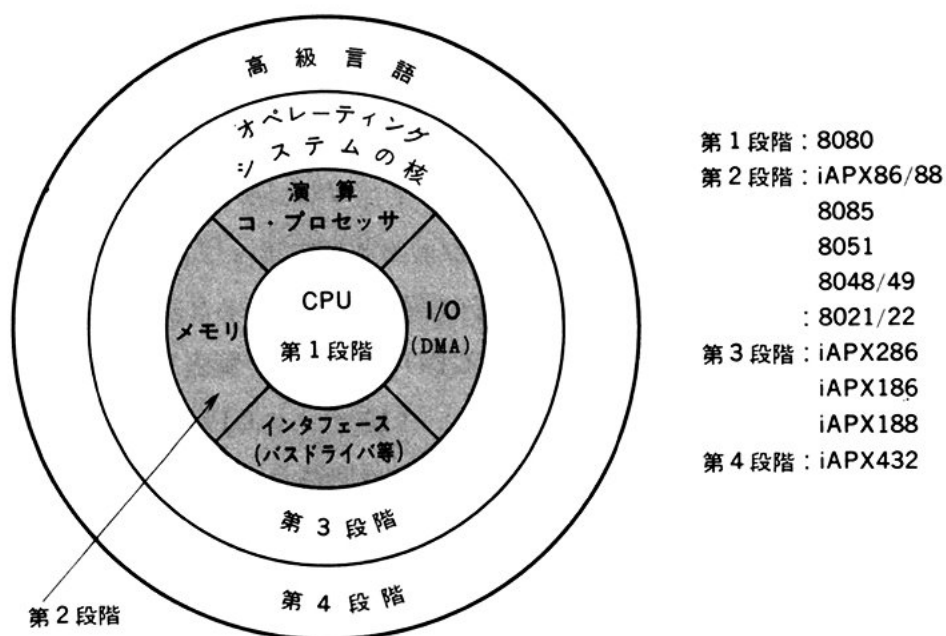



図 12・7　マイクロシステム80の階層的分類

表 12・5　マイクロシステムの分類

マイクロシステムクラス名	機能レベル	相対性能		メモリ		代表的製品
		CPU	I/O	プログラムサイズ	メモリマネジメント	
マイクロメインフレーム	32ビット	20～70	2～15	256K～8M		iAPX 432
マイクロマキシ	16/32ビット	25	4	128K～1M		iAPX 286
マイクロミディ	16ビット	8～10	2	32K～256K		iAPX 86
マイクロコンピュータ	8/16ビット	1～5	1～1.5	16K～120K		iAPX 88
マイクロコントローラ	8ビット	1	0.3	4K～32K		8048 8051 8085

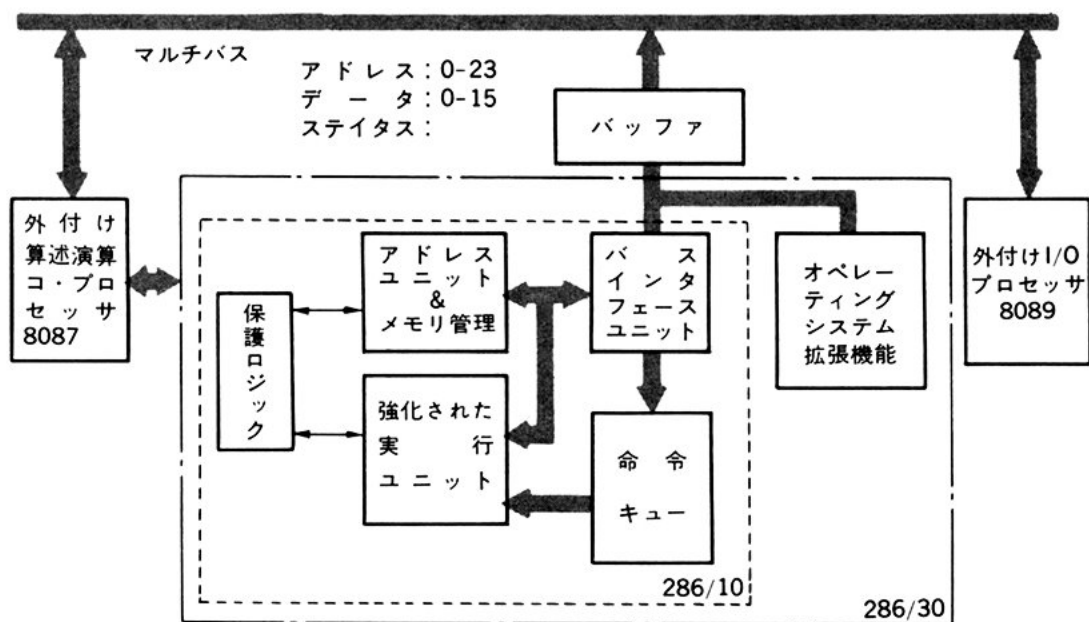



図 12・8　iAPX286/10 および iAPX286/30

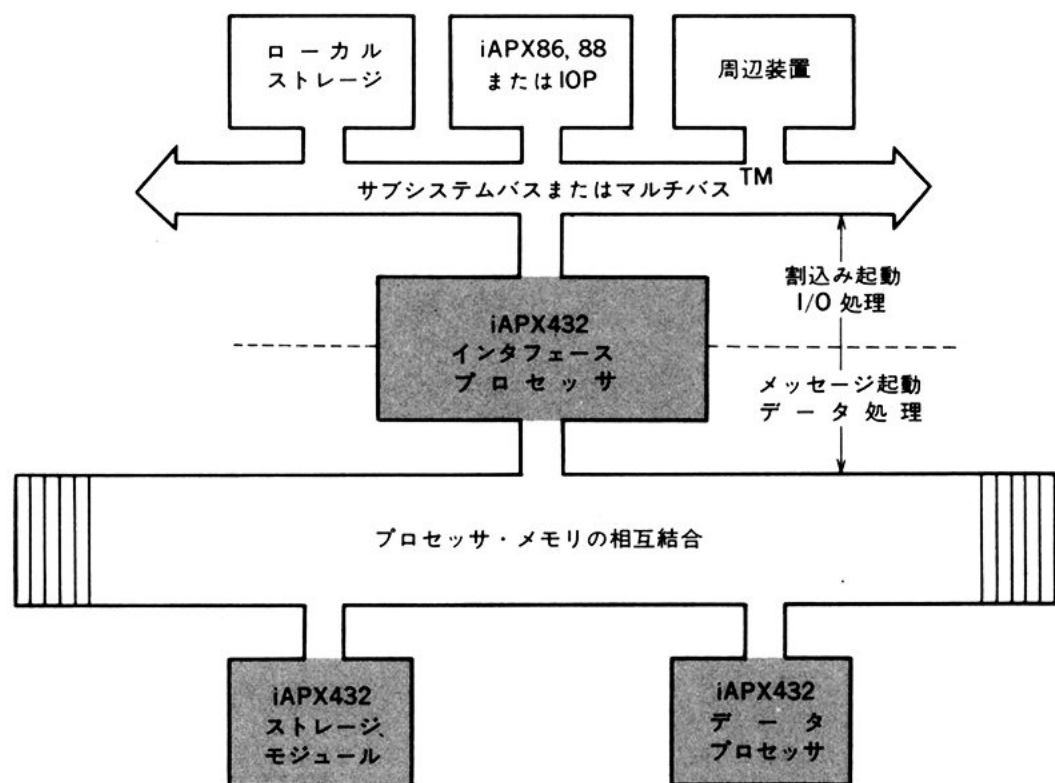



図 12・9　iAPX432 システムの構成

付　　録

付録 I. ASM 86 プログラム例

```
MCS-86 MACRO ASSEMBLER    DICE

ISIS-II MCS-86 MACRO ASSEMBLER V2.0 ASSEMBLY OF MODULE DICE
OBJECT MODULE PLACED IN :F1:DICE.OBJ
ASSEMBLER INVOKED BY: ASM86 :F1:DICE.A86 XREF

LOC  OBJ                  LINE     SOURCE

                            1      ; THIS PROGRAM SIMULATES THE ROLL OF A PAIR OF DICE
                            2
                            3      ; CONSOLE OUTPUT PROCEDURE
                            4              EXTRN   CO:NEAR
                            5
                            6      ; SEGMENT GROUP DEFINITIONS NEEDED FOR PL/M-86 COMPATIBILITY
                            7      CGROUP  GROUP   CODE
                            8      DGROUP  GROUP   DATA,STACK
                            9
                           10      ; INFORM ASSEMBLER OF SEGMENT REGISTER CONTENTS.
                           11              ASSUME  CS:CGROUP,DS:DGROUP,SS:DGROUP,ES:NOTHING
                           12
                           13      ; ALLOCATE DATA
----                       14      DATA    SEGMENT PUBLIC  'DATA'
                           15      ; NOTE THAT THE FOLLOWING ARE PASSED ON THE STACK TO THE PL/M-86
                           16      ;  PROCEDURE 'CO'.  BY CONVENTION, A BYTE PARAMETER IS PASSED IN
                           17      ;  THE LOW-ORDER 8-BITS OF A WORD ON THE STACK.  HENCE, THESE ARE
                           18      ;  DEFINED AS WORD VALUES, THOUGH THEY OCCUPY 1 BYTE ONLY.
0000 1B00                  19      CLEAR_CRT1      DW      01BH    ; INTELLEC
0002 4500                  20      CLEAR_CRT2      DW      045H    ;   CRT
0004 1B00                  21      HOME_CURSOR1    DW      01BH    ;   CONTROL
0006 4800                  22      HOME_CURSOR2    DW      048H    ;   CODES
0008 2000                  23      SPACE           DW      020H    ; ASCII BLANK
000A ????                  24      SAVE            DW      ?       ; HOLDS LAST 16-BIT RANDOM NUMBER
----                       25      DATA    ENDS
                           26
                           27
                           28      ; ALLOCATE STACK SPACE
----                       29      STACK   SEGMENT STACK   'STACK'
0000 (20                   30              DW      20 DUP (?)
     ????
     )
                           31      ; LABEL INITIAL TOS: FOR LATER USE.
0028                       32      STACK_TOP       LABEL   WORD
----                       33      STACK   ENDS
                           34
                           35
                           36      ; PROGRAM CODE
----                       37      CODE    SEGMENT PUBLIC  'CODE'
                           38
                           39
                           40      ; RANDOM NUMBER GENERATOR PROCEDURE
                           41      ; ALGORITHM FOR 16-BIT RANDOM NUMBER FROM:
                           42      ;   "A GUIDE TO PL/M PROGRAMMING FOR
                           43      ;   MICROCOMPUTER APPLICATIONS,"
                           44      ;   DANIEL D. MCCRACKEN
                           45      ;   ADDISON-WESLEY, 1978
0000                       46      RANDOM  PROC
0000 A10A00          R     47              MOV     AX,SAVE         ; NEW NUMBER =
0003 B90508                48              MOV     CX,2053         ;   OLD NUMBER * 2053
0006 F7E1                  49              MUL     CX              ;   + 13849
0008 051936                50              ADD     AX,13849        ;
000B A30A00          R     51              MOV     SAVE,AX         ; SAVE FOR NEXT TIME
                           52              ; FORCE 16-BIT NUMBER INTO RANGE 1 - 6
                           53              ;   BY MODULO 6 DIVISION + 1
000E 2BD2                  54              SUB     DX,DX           ; CLEAR UPPER DIVIDEND
0010 B90600                55              MOV     CX,6            ; SET DIVISOR
0013 F7F1                  56              DIV     CX              ; DIVIDE BY 6
0015 8BC2                  57              MOV     AX,DX           ; REMAINDER TO AX
0017 40                    58              INC     AX              ; ADD 1
0018 C3                    59              RET                     ; RESULT IN AX
                           60      RANDOM  ENDP
                           61
```

```
                        62
                        63      ; MAIN PROGRAM
                        64
                        65      ; LOAD SEGMENT REGISTERS
                        66      ; NOTE PROGRAM DOES NOT USE ES; CS IS INITIALIZED BY HARDWARE RESET;
                        67      ;  DATA & STACK ARE MEMBERS OF SAME GROUP, SO ARE TREATED AS A SINGLE
                        68      ;  MEMORY SEGMENT POINTED TO BY BOTH DS & SS.
0019 B8----        R    69      START:  MOV     AX,DGROUP
001C 8ED8               70              MOV     DS,AX
001E 8ED0               71              MOV     SS,AX
                        72
                        73      ; INITIALIZE STACK POINTER
0020 BC2800        R    74              MOV     SP,OFFSET DGROUP:STACK_TOP
                        75
                        76      ; CLEAR THE SCREEN
0023 FF36000C      R    77              PUSH    CLEAR_CRT1
0027 E80000        E    78              CALL    CO
002A FF360200      R    79              PUSH    CLEAR_CRT2
002E E80000        E    80              CALL    CO
                        81
                        82      ; ROLL THE DICE UNTIL INTERRUPTED
0031 E8CCFF             83      ROLL:   CALL    RANDOM          ; GET 1ST DIE IN AL
0034 0430               84              ADD     AL,030H         ; CONVERT TO ASCII
0036 50                 85              PUSH    AX              ; PASS IT TO
0037 E80000        E    86              CALL    CO              ;   CONSOLE OUTPUT
003A FF360800      R    87              PUSH    SPACE           ; OUTPUT
003E E80000        E    88              CALL    CO              ;   A BLANK
0041 E8BCFF             89              CALL    RANDOM          ; GET 2ND DIE IN AL
0044 0430               90              ADD     AL,030H         ; CONVERT TO ASCII
0046 50                 91              PUSH    AX              ; PASS IT TO
0047 E80000        E    92              CALL    CO              ;   CONSOLE OUTPUT
                        93      ; HOME THE CURSOR
004A FF360400      R    94              PUSH    HOME_CURSOR1
004E E80000        E    95              CALL    CO
0051 FF360600      R    96              PUSH    HOME_CURSOR2
0055 E80000        E    97              CALL    CO
                        98      ; CONTINUE FOREVER
0058 EBD7               99              JMP     ROLL
----                   100      CODE    ENDS
                       101
```

付録2. PL/M-86 プログラム例

```
PL/M-86 COMPILER    DICE

ISIS-II PL/M-86 V1.2 COMPILATION OF MODULE DICE
OBJECT MODULE PLACED IN :F1:DICE.OBJ
COMPILER INVOKED BY:  PLM86 :F1:DICE.P86 XREF

   1          DICE:  DO;
              /* THIS PROGRAM SIMULATES THE ROLL OF A PAIR OF DICE */

              /* GIVE NAMES TO CONSTANTS */
   2   1      DECLARE CLEAR$CRT1       LITERALLY '01BH';  /* INTELLEC   */
   3   1      DECLARE CLEAR$CRT2       LITERALLY '045H';  /*  CRT       */
   4   1      DECLARE HOME$CURSOR1     LITERALLY '01BH';  /*  CONTROL   */
   5   1      DECLARE HOME$CURSOR2     LITERALLY '048H';  /*  CODES     */
   6   1      DECLARE SPACE            LITERALLY '020H';  /*ASCII BLANK*/

              /* PROGRAM VARIABLES */
   7   1      DECLARE (RANDOM$NUMBER,SAVE)  WORD;

              /* CONSOLE OUTPUT PROCEDURE */
   8   1      CO:  PROCEDURE(X) EXTERNAL;
   9   2           DECLARE X    BYTE;
  10   2           END CO;

              /* RANDOM NUMBER GENERATOR PROCEDURE                  */
              /* ALGORITHM FOR 16-BIT RANDOM NUMBER FROM:           */
              /*    "A GUIDE TO PL/M PROGRAMMING FOR                */
              /*     MICROCOMPUTER APPLICATIONS,"                   */
              /*    DANIEL D. MCCRACKEN,                            */
              /*    ADDISON-WESLEY, 1978                            */
  11   1      RANDOM:  PROCEDURE WORD;
  12   2         RANDOM$NUMBER = SAVE;           /*START WITH OLD NUMBER*/
  13   2         RANDOM$NUMBER = 2053 * RANDOM$NUMBER + 13849;
  14   2         SAVE = RANDOM$NUMBER;           /*SAVE FOR NEXT TIME*/
                 /*FORCE 16-BIT NUMBER INTO RANGE 1-6*/
  15   2         RANDOM$NUMBER = RANDOM$NUMBER MOD 6  + 1;
  16   2         RETURN RANDOM$NUMBER;
  17   2         END RANDOM;

              /* MAIN ROUTINE */
              /* CLEAR THE SCREEN*/
  18   1      CALL CO(CLEAR$CRT1);
  19   1      CALL CO(CLEAR$CRT2);

              /* ROLL THE DICE UNTIL INTERRUPTED */
  20   1      DO WHILE 1;    /*"DO FOREVER"*/
                 /*NOTE THAT ADDING 30 TO THE DIE VALUE */
                 /*  CONVERTS IT TO ASCII.              */
  21   2         CALL CO(RANDOM + 030H);        /*1ST DIE*/
  22   2         CALL CO(SPACE);                /*BLANK*/
  23   2         CALL CO(RANDOM + 030H);        /*2ND DIE*/
                 /* HOME THE CURSOR */
  24   2         CALL CO(HOME$CURSOR1);
  25   2         CALL CO(HOME$CURSOR2);
  26   2         END;

  27   1      END DICE;
```

付録3．8086/8088命令一覧

データ転送

MOV=Move:

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタ/メモリ↔レジスタ	100010dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接→レジスタ/メモリ	1100011w	mod 000 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
直接→レジスタ	1011w reg	data	data if w=1			
メモリ→アキュムレータ	1010000w	addr-lo	addr-hi			
アキュムレータ→メモリ	1010001w	addr-lo	addr-hi			
レジスタ/メモリ→セグメントレジスタ	10001110	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		
セグメントレジスタ→レジスタ/メモリ	10001100	mod 0 SR r/m	(DISP-LO)	(DISP-HI)		

PUSH=Push:

	76543210	76543210	76543210	76543210
レジスタ/メモリ	11111111	mod 110 r/m	(DISP-LO)	(DISP-HI)
レジスタ	01010 reg			
セグメントレジスタ	000 reg 110			

POP=Pop:

	76543210	76543210	76543210	76543210
レジスタ/メモリ	10001111	mod 000 r/m	(DISP-LO)	(DISP-HI)
レジスタ	01011 reg			
セグメントレジスタ	000 reg 111			

XCHG＝Exchange :

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタとレジスタ/メモリ	1000011w	mod reg r/m	(DISP-LO)	(DISP-HI)		
アキュムレータとレジスタ	10010 reg					

IN＝Input from :

固定ポート	1110010w	DATA-8
可変ポート	1110110w	

OUT＝Output to :

固定ポート	1110011w	DATA-8		
可変ポート	1110111w			
XLAT ＝バイト翻訳→AL	11010111			
LEA ＝EA→レジスタ	10001101	mod reg r/m	(DISP-LO)	(DISP-HI)
LDS ＝ポインタ→DS	11000101	mod reg r/m	(DISP-LO)	(DISP-HI)
LES ＝ポインタ→ES	11000100	mod reg r/m	(DISP-LO)	(DISP-HI)
LAHF ＝AH←フラグ	10011111			
SAHF ＝AH→フラグ	10011110			
PUSHF＝フラグをプッシュ	10011100			
POPF ＝フラグをポップ	10011101			

算術命令

ADD＝Add：

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタとレジスタ/メモリ相互	000000dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接→レジスタ/メモリ	100000sw	mod 000 r/m	(DISP-LO)	(DISP-HI)	data	data if s:w=01
直接→アキュムレータ	0000010w	data	data if w=1			

ADC＝Add with carry：

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタとレジスタ/メモリ相互	000100dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接→レジスタ/メモリ	100000sw	mod 010 r/m	(DISP-LO)	(DISP-HI)	data	data if s:w=01
直接→アキュムレータ	0001010w	data	data if w=1			

INC＝Increment：

	76543210	76543210	76543210	76543210
レジスタ/メモリ	1111111w	mod 000 r/m	(DISP-LO)	(DISP-HI)
レジスタ	01000 reg			

	76543210
AAA＝加算のためのASCII調整	00110111
DAA＝加算のための10進調整	00100111

SUB＝Subtract：

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタとレジスタ/メモリ相互	001010dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
レジスタ/メモリから直接	100000sw	mod 101 r/m	(DISP-LO)	(DISP-HI)	data	data if s:w=01
アキュムレータから直接	0010110w	data	data if w=1			

	76543210	76543210	76543210	76543210	76543210	76543210
SBB=Subtract with borrow:						
レジスタとレジスタ/メモリ相互	000110dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
レジスタ/メモリから直接	100000sw	mod 011 r/m	(DISP-LO)	(DISP-HI)	data	data if s:w=01
アキュムレータから直接	0001110w	data	data if w=1			
DEC=Decrement:						
レジスタ/メモリ	1111111w	mod 001 r/m	(DISP-LO)	(DISP-HI)		
レジスタ	01001 reg					
NEG=符号変換	1111011w	mod 011 r/m	(DISP-LO)	(DISP-HI)		
CMP=Compare:						
レジスタとレジスタ/メモリ	001110dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
レジスタ/メモリと直接	100000sw	mod 111 r/m	(DISP-LO)	(DISP-HI)	data	data if s:w=1
アキュムレータと直接	0011110w	data				
AAS =減算のためのASCII調整	00111111					
DAS =減算のための10進調整	00101111					
MUL =乗算(符号なし)	1111011w	mod 100 r/m	(DISP-LO)	(DISP-HI)		
IMUL=整数乗算(符号付)	1111011w	mod 101 r/m	(DISP-LO)	(DISP-HI)		
AAM =乗算のためのASCII調整	11010100	00001010	(DISP-LO)	(DISP-HI)		
DIV =除算(符号なし)	1111011w	mod 110 r/m	(DISP-LO)	(DISP-HI)		
IDIV=整数除算(符号付)	1111011w	mod 111 r/m	(DISP-LO)	(DISP-HI)		
AAD =除算のためのASCII調整	11010101	00001010	(DISP-LO)	(DISP-HI)		

命令	76543210	76543210	76543210	76543210	76543210	76543210
CSW=バイト→ワード変換	10011000					
CWD=ワード→ダブルワード変換	10011001					

論理命令

命令				
NOT=invert	1111011w	mod 010 r/m	(DISP-LO)	(DISP-HI)
SHL/SAL=論理/算術左シフト	110100vw	mod 100 r/m	(DISP-LO)	(DISP-HI)
SHR=論理右シフト	110100vw	mod 101 r/m	(DISP-LO)	(DISP-HI)
SAR=算術右シフト	110100vw	mod 111 r/m	(DISP-LO)	(DISP-HI)
ROL=左回転	110100vw	mod 000 r/m	(DISP-LO)	(DISP-HI)
ROR=右回転	110100vw	mod 001 r/m	(DISP-LO)	(DISP-HI)
RCL=キャリーを含め左回転	110100vw	mod 010 r/m	(DISP-LO)	(DISP-HI)
RCR=キャリーを含め右回転	110100vw	mod 011 r/m	(DISP-LO)	(DISP-HI)

AND=And:

レジスタとレジスタ/メモリ相互	001000dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
レジスタ/メモリと直接値	1000000w	mod 100 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
アキュムレータと直接値	0010010w	data	data if w=1			

TEST=フラグに対するAND機能で，結果なし:

レジスタ/メモリとレジスタ	000100dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接データとレジスタ/メモリ	1111011w	mod 000 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
直接データとアキュムレータ	1010100w	data				

OR=Or:

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタ/メモリとレジスタ相互	000010dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接値とレジスタメモリ	1000000w	mod 001 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
直接値とアキュムレータ	0000110w	data	data if w=1			

XOR=Exclusive or:

	76543210	76543210	76543210	76543210	76543210	76543210
レジスタ/メモリとレジスタ相互	001100dw	mod reg r/m	(DISP-LO)	(DISP-HI)		
直接値とレジスタ/メモリ	1000000w	mod 110 r/m	(DISP-LO)	(DISP-HI)	data	data if w=1
直接値とアキュムレータ	0011010w	data	data if w=1			

ストリング操作

	76543210
REP =繰返し	1111001z
MOVS=バイト/ワード移動	1010010w
CMPS=バイト/ワード比較	1010011w
SCAS=バイト/ワード走査	1010111w
LODS=バイト/ワードをAL/AXにロード	1010110w
STDS=AL/AXからバイト/ワードをストア	1010101w

コントロール転送

CALL＝Call：

	76543210	76543210	76543210	76543210	76543210	76543210
セグメント内直接	11101000	IP-INC-LO	IP-INC-HI			
セグメント内間接	11111111	mod 010 r/m	(DISP-LO)	(DISP-HI)		
セグメント外直接	10011010	IP-lo	IP-hi			
		CS-lo	CS-hi			
セグメント外間接	11111111	mod 011 r/m	(DISP-LO)	(DISP-HI)		

JMP＝無条件ジャンプ

セグメント内直接	11101001	IP-INC-LO	IP-INC-HI	
セグメントショート内直接	11101011	IP-INC8		
セグメント内間接	11111111	mod 100 r/m	(DISP-LO)	(DISP-HI)
セグメント外直接	11101010	IP-lo	IP-hi	
		CS-lo	CS-hi	
セグメント外間接	11111111	mod 101 r/m	(DISP-LO)	(DISP-HI)

RET＝CALL からの復帰

セグメント内	11000011		
直接値を SP に加算したセグメント内	11000010	data-lo	data-hi
セグメント外	11001011		
直接値を SP に加算したセグメント外	11001010	data-lo	data-hi

	76543210	76543210	76543210	76543210	76543210	76543210
JE/JZ =等しい/ゼロでジャンプ	01110100	IP-INC8				
JL/JNGE=以下/大きくないまたは等しいでジャンプ	01111100	IP-INC8				
JLE/JNG=以下または等しい/大きくないでジャンプ	01111110	IP-INC8				
JB/JNAE=下/上でないまたは等しいでジャンプ	01110010	IP-INC8				
JBE/JNA=下または等しい/上でないでジャンプ	01110110	IP-INC8				
JP/JPE =パリティ/パリティ偶数でジャンプ	01111010	IP-INC8				
JO =オーバフローでジャンプ	01110000	IP-INC8				
JS =符号でジャンプ	01111000	IP-INC8				
JNE/JNZ=等しくない/ゼロでないでジャンプ	01110101	IP-INC8				
JNL/JNG=以下でない/大きくないでジャンプ	01111101	IP-INC8				
JNLE/JG=以下でないまたは等しい/大きいでジャンプ	01111111	IP-INC8				
JNB/JAE=下でない/上または等しいでジャンプ	01110011	IP-INC8				
JNBE/JA=下でないまたは等しい/上でジャンプ	01110111	IP-INC8				
JNP/JPO=パリティなし/パリティ奇数でジャンプ	01111011	IP-INC8				
JNO =オーバフローでないジャンプ	01110001	IP-INC8				
JNS =符号なしでジャンプ	01111001	IP-INC8				
LOOP =CX回ループ	11100010	IP-INC8				
LOOPZ/LOOPE=ゼロ/等しい間ループ	11100001	IP-INC8				
LOOPZ/LOOPNE=ゼロでない/等しくない間ループ	11100000	IP-INC8				
JCXZ =CXゼロでジャンプ	11100011	IP-INC8				

INT=Interrupt:

	76543210	76543210	76543210	76543210	76543210	76543210
指定タイプ	11001101	DATA-8				
タイプ 3	11001100					
オーバフローで割込み	11001110					
割込み復帰	11001111					

プロセッサコントロール

CLC =キャリークリヤ	11111000			
CMC =キャリーの補数	11110101			
STC =キャリーセット	11111001			
CLD =ディレクションクリヤ	11111100			
STD =ディレクションをセット	11111101			
CLI =割込みクリヤ	11111010			
STI =割込みセット	11111011			
HLT =停止(ホルト)	11110100			
WAIT=ウエイト	10011011			
ESC =エスケープ(外部装置に対し)	11011xxx	mod yyy r/m	(DISP-LO)	(DISP-HI)
LOCK=バスロックプリフィックス	11110000			
SEGMENT=オーバライドプリフィックス	001reg110			

付録4．8086/8088命令のマトリックス一覧（Mnemonics © Intel，1978）

Hi＼Lo	0	1	2	3	4	5	6	7	8	9	A	B	C	D	E	F
0	ADD b.f.r/m	ADD w.f.r/m	ADD b.t.r/m	ADD w.t.r/m	ADD b.ia	ADD w.ia	PUSH ES	POP ES	OR b.f.r/m	OR w.f.r/m	OR b.t.r/m	OR w.t.r/m	OR b.i	OR w.i	PUSH CS	
1	ADC b.f.r/m	ADC w.f.r/m	ADC b.t.r/m	ADC w.t.r/m	ADC b.i	ADC w.i	PUSH SS	POP SS	SBB b.f.r/m	SBB w.f.r/m	SBB b.t.r/m	SBB w.t.r/m	SBB b.i	SBB w.i	PUSH DS	POP DS
2	AND b.f.r/m	AND w.f.r/m	AND b.t.r/m	AND w.t.r/m	AND b.i	AND w.i	SEG =ES	DAA	SUB b.f.r/m	SUB w.f.r/m	SUB b.t.r/m	SUB w.t.r/m	SUB b.i	SUB w.i	SEG =CS	DAS
3	XOR b.f.r/m	XOR w.f.r/m	XOR b.t.r/m	XOR w.t.r/m	XOR b.i	XOR w.i	SEG =SS	AAA	CMP b.f.r/m	CMP w.f.r/m	CMP b.t.r/m	CMP w.t.r/m	CMP b.i	CMP w.i	SEG =DS	AAS
4	INC AX	INC CX	INC DX	INC BX	INC SP	INC BP	INC SI	INC DI	DEC AX	DEC CX	DEC DX	DEC BX	DEC SP	DEC BP	DEC SI	DEC DI
5	PUSH AX	PUSH CX	PUSH DX	PUSH BX	PUSH SP	PUSH BP	PUSH SI	PUSH DI	POP AX	POP CX	POP DX	POP BX	POP SP	POP BP	POP SI	POP DI
6																
7	JO	JNO	JB/ JNAE	JNB/ JAE	JE/ JZ	JNE/ JNZ	JBE/ JNA	JNBE/ JA	JS	JNS	JP/ JPE	JNP/ JPO	JL/ JNGE	JNL/ JGE	JLE/ JNG	JNLE/ JG
8	lmmed b.r/m	lmmed w.r/m	lmmed b.r/m	lmmed is.r/m	TEST b.r/m	TEST w.r/m	XCHG b.r/m	XCHG w.r/m	MOV b.f.r/m	MOV w.f.r/m	MOV b.t.r/m	MOV w.t.r/m	MOV sr.f.r/m	LEA	MOV sr.t.r/m	POP r/m
9	XCHG AX	XCHG CX	XCHG DX	XCHG BX	XCHG SP	XCHG BP	XCHG SI	XCHG DI	CBW	CWD	CALL l.d	WAIT	PUSHF	POPF	SAHF	LAHF
A	MOV m→AL	MOV m→AX	MOV AL→m	MOV AX→m	MOVS	MOVS	CMPS	CMPS	TEST b.i,a	TEST w.i,a	STOS	STOS	LODS	LODS	SCAS	SCAS
B	MOV i→AL	MOV i→CL	MOV i→DL	MOV i→BL	MOV i→AH	MOV i→CH	MOV i→DH	MOV i→BH	MOV i→AX	MOV i→CX	MOV i→DX	MOV i→BX	MOV i→SP	MOV i→BP	MOV i→SI	MOV i→DI
C			RET (i+SP)	RET	LES	LDS	MOV b.i.r/m	MOV w.i.r/m			RET l.(i+SP)	RET l	INT Type 3	INT (Any)	INTO	IRET
D	Shift b	Shift w	Shift b.v	Shift w.v	AAM	AAD		XLAT	ESC 0	ESC 1	ESC 2	ESC 3	ESC 4	ESC 5	ESC 6	ESC 7
E	LOOPNZ/ LOOPNE	LOOPZ/ LOOPE	LOOP	JCXZ	IN b	IN w	OUT b	OUT w	CALL d	JMP d	JMP I.d	JMP si.d	IN v,b	IN v,w	OUT v,b	OUT v.w
F	LOCK		REP	REP z	HLT	CMC	Grp1 b.r/m	Grp1 w.r/m	CLC	STC	CLI	STI	CLD	STD	Grp 2 b.r/m	Grp 2 w.r/m

［注］ b：バイト動作　d：直接(direct)　f：CPUレジスタから　i：直接(immediate)
ia：アキュムレータへの直接データ　id：間接　is：符号拡張された直接バイト
l：セグメント外，ロング　m：メモリ　r/m：EAが2番目のバイトになる
Si：セグメント内，ショート　Sr：セグメントレジスタ　t：CPUレジスタへ
w：ワード動作　v：変数　z：ゼロ

付録5．8086/8088の電気的特性

〔1〕 最大定格

- バイアス下の周囲温度　0～70°C
- 保存温度　－65～＋150°C
- GND に対する任意のピンの電圧　－1.0～＋7 V
- 消費電力　2.5 W

〔2〕 直流特性

8086　：T_A＝0～70°C，V_{CC}＝5 V±10％

8086-1：T_A＝0～70°C，V_{CC}＝5 V± 5％

8086-2：T_A＝0～70°C，V_{CC}＝5 V± 5％

記号	定数名	最小	最大	単位	試験条件
V_{1L}	入力低電圧	－0.5	＋0.8	V	
V_{1H}	入力高電圧	2.0	V_{CC}＋0.5	V	
V_{OL}	出力低電圧		0.45	V	I_{OL}＝2.5 mA
V_{OH}	出力高電圧	2.4		V	I_{OH}＝－400 μA
I_{CC}	供給電源電圧　8086 8086-1 8086-2		340 360 350	mA	T_A＝25 °C
I_{L1}	入力漏れ電流		±10	μA	0 V ≦ V_{IN} ≦ V_{CC}
I_{L0}	出力漏れ電流		±10	μA	0.45 V ≦ V_{OUT} = V_{CC}
V_{CL}	クロック入力低電圧	－0.5	＋0.6	V	
V_{CH}	クロック入力高電圧	3.9	V_{CC}＋1.0	V	
C_{IN}	入力バッファ容量 （AD_0-AD_{15}, $\overline{RQ}/\overline{GT}$ を除くすべての入力）		15	pF	f_C＝1 MHz
C_{10}	I/Oバッファ容量 （AD_0-A_{15}，$\overline{RQ}/\overline{GT}$）		15	pF	f_C＝1 MHz

〔3〕 **交流特性(1)** (8086ミニマムシステム)

8086　：T_A＝0～70°C,　V_{CC}＝5V±10％

8086-1：T_A＝0～70°C,　V_{CC}＝5V±5％

8086-2：T_A＝0～70°C,　V_{CC}＝5V±5％

〈タイミング規格〉

記号	定数名	8086		8086-1(予備)		8086-2		単位	試験条件
		最小	最大	最小	最大	最小	最大		
TCLCL	クロック周期	200	500	100	500	125	500	ns	
TCLCH	CLK低時間	118		53		68		ns	
TCHCL	CLK高時間	69		39		44		ns	
TCH1CH2	CLK立上り時間		10		10		10	ns	1.0～3.5V
TCL2CL1	CLK立下り時間		10		10		10	ns	3.5～1.0V
TDVCL	データ入力設定時間	30		5		20		ns	
TCLDX	データ入力保持時間	10		10		10		ns	
TR1VCL	8284AへのRDY設定時間(注1, 2)	35		35		35		ns	
TCLR1X	8284AへのRDY保持時間(注1, 2)	0		0		0		ns	
TRYHCH	8086へのREADY設定時間	118		53		68		ns	
TCHRYX	8086へのREADY保持時間	30		20		20		ns	
TRYLCL	READY不活性からCLK(注3)	−8		−10		−8		ns	
THVCH	HOLD設定時間	35		20		20		ns	
TINVCH	INTR, NMI, $\overline{\text{TEST}}$設定時間(注2)	30		15		15		ns	
TILIH	入力立上り時間(CLK除く)		20		20		20	ns	0.8～2.0V
TIHIL	入力立下り時間(CLK除く)		12		12		12	ns	2.0～0.8V

〔注〕 1. 参考のために8284Aの信号を示した.
2. 次のクロックでの認識を保証するための非同期信号の設定要求.
3. T_2ステートのみに適用(T_3内へ8ns).

〈タイミング応答〉

記号	定数名	8086		8086-1(予備)		8086-2		単位	試験条件
		最小	最大	最小	最大	最小	最大		
TCLAV	アドレス有効遅延	10	110	10	50	10	60	ns	すべての8086出力に対し C_L = 20−100 pF (8086自身の負荷効果に加えて)
TCLAX	アドレスホールド時間	10		10		10		ns	
TCLAZ	アドレスフロート遅延	TCLAX	80	10	40	TCLAX	50	ns	
TLHLL	ALE幅	TCLCH -20		TCLCH -10		TCLCH -10		ns	
TCLLH	ALE活性遅延		80		40		50	ns	
TCHLL	ALE不活性遅延		85		45		55	ns	
TLLAX	アドレスホールド時間からALE不活性まで	TCHCL -10		TCHCL -10		TCHCL -10		ns	
TCLDV	データ有効遅延	10	110	10	50	10	60	ns	
TCHDX	データホールド時間	10		10		10		ns	
TWHDX	WR後のデータホールド時間	TCLCH -30		TCLCH -25		TCLCH -30		ns	
TCVCTV	コントロール活性遅延1	10	110	10	50	10	70	ns	
TCHCTV	コントロール活性遅延2	10	110	10	45	10	60	ns	
TCVCTX	コントロール不活性遅延	10	110	10	50	10	70	ns	
TAZRL	アドレスフロートからREAD活性	0		0		0		ns	
TCLRL	$\overline{RD}$活性遅延	10	165	10	70	10	100	ns	
TCLRH	$\overline{RD}$不活性遅延	10	150	10	60	10	80	ns	
TRHAV	$\overline{RD}$不活性から次のアドレス活性	TCLCL -45		TCLCL -35		TCLCL -40		ns	
TCLHAV	HOLDA有効遅延	10	160	10	60	10	100	ns	
TRLRH	$\overline{RD}$幅	2TCLCL -75		2TCLCL -40		2TCLCL -50		ns	
TWLWH	$\overline{WR}$幅	2TCLCL -60		2TCLCL -35		2TCLCL -40		ns	
TAVAL	アドレス有効からALE低	TCLCH -60		TCLCH -35		TCLCH -40		ns	
TOLOH	出力立上り時間		20		20		20	ns	0.8〜2.0V
TOHOL	出力立下り時間		12		12		12	ns	2.0〜0.8V

〔4〕 交流特性(2) 〔8086マキシマムモード(8288バスコントローラ使用)〕

〈タイミング規格〉

記号	定数名	8086		8086-1(予備)		8086-2(予備)		単位	試験条件
		最小	最大	最小	最大	最小	最大		
TCLCL	クロック周期	200	500	100	500	125	500	ns	
TCLCH	CLK低時間	118		53		68		ns	
TCHCL	CLK高時間	69		39		44		ns	
TCH1CH2	CLK立上り時間		10		10		10	ns	1.0～3.5V
TCL2CL1	CLK立下り時間		10		10		10	ns	3.5～1.0V
TDVCL	データ入力設定時間	30		5		20		ns	
TCLDX	データ入力ホールド時間	10		10		10		ns	
TR1VCL	8284AへのRDY設定時間(注1, 2)	35		35		35		ns	
TCLR1X	8284AへのRDYホールド時間(注1, 2)	0		0		0		ns	
TRYHCH	8086へのREADY設定時間	118		53		68		ns	
TCHRYX	8086へのREADYホールド時間	30		20		20		ns	
TRYLCL	READY不活性からCLKまで(注4)	−8		−10		−8		ns	
TINVCH	認識のための設定時間(注2)(INTR, NMI, $\overline{\text{TEST}}$)	30		15		15		ns	
TGVCH	$\overline{\text{RQ}}/\overline{\text{GT}}$設定時間	30		12		15		ns	
TCHGX	8086への$\overline{\text{RQ}}$ホールド時間	40		20		30		ns	
TILIH	入力立上り時間(CLK除く)		20		20		20	ns	0.8～2.0V
TIHIL	入力立下り時間(CLK除く)		12		12		12	ns	2.0～0.8V

〈タイミング応答〉

記号	定数名	8086		8086-1(予備)		8086-2(予備)		単位	試験条件
		最小	最大	最小	最大	最小	最大		
TCLML	コマンド活性遅延(注1)	10	35	10	35	10	35	ns	すべての8086出力に対して C_L = 20－100 pF (8086自身の負荷効果を加えて)
TCLMH	コマンド不活性遅延(注1)	10	35	10	35	10	35	ns	
TRYHSH	READY 活性からステイタス受動へ(注3)		110		45		65	ns	
TCHSV	ステイタス活性遅延	10	110	10	45	10	60	ns	
TCLSH	ステイタス不活性遅延	10	130	10	55	10	70	ns	
TCLAV	アドレス有効遅延	10	110	10	50	10	60	ns	
TCLAX	アドレスホールド時間	10		10		10		ns	
TCLAZ	アドレスフロート遅延	TCLAX	80	10	40	TCLAX	50	ns	
TSVLH	ステイタス有効からALE高(注1)		15		15		15	ns	
TSVMCH	ステイタス有効からMCE高(注1)		15		15		15	ns	
TCLLH	CLK低からALE有効まで(注1)		15		15		15	ns	
TCLMCH	CLK低からMCE高まで(注1)		15		15		15	ns	
TCHLL	ALE不活性遅延(注1)		15		15		15	ns	
TCLMCL	MCE不活性遅延(注1)		15		15		15	ns	
TCLDV	データ有効遅延	10	110	10	50	10	60	ns	
TCHDX	データホールド時間	10		10		10		ns	
TCVNV	コントロール活性遅延(注1)	5	45	5	45	5	45	ns	
TCVNX	コントロール不活性遅延(注1)	10	45	10	45	10	45	ns	
TAZRL	アドレスフロートからリード活性まで	0		0		0		ns	
TCLRL	RD活性遅延	10	165	10	70	10	100	ns	
TCLRH	RD不活性遅延	10	150	10	60	10	80	ns	
TRHAV	RD不活性から次のアドレス活性	TCLCL -45		TCLCL -35		TCLCL -40		ns	

(次ページへつづく)

（前ページよりのつづき）

記　号	定　数　名	8086		8086-1（予備）		8086-2（予備）		単位	試験条件
		最小	最大	最小	最大	最小	最大		
TCHDTL	方向コントロール活性遅延（注1）		50		50		50	ns	すべての8086出力に対して $C_L=$ 20－100pF（8086自身の負荷効果を加えて）
TCHDTH	方向コントロール不活性遅延（注1）		30		30		30	ns	
TCLGL	$\overline{\mathrm{GT}}$ 活性遅延	0	85	0	45	0	50	ns	
TCLGH	$\overline{\mathrm{GT}}$ 不活性遅延	0	85	0	45	0	50	ns	
TRLRH	$\overline{\mathrm{RD}}$ 幅	2TCLCL -75		2TCLCL -40		2TCLCL -50		ns	
TOLOH	出力立上り時間		20		20		20	ns	0.8→2.0 V
TOHOL	出力立下り時間		12		12		12	ns	2.0→0.8 V

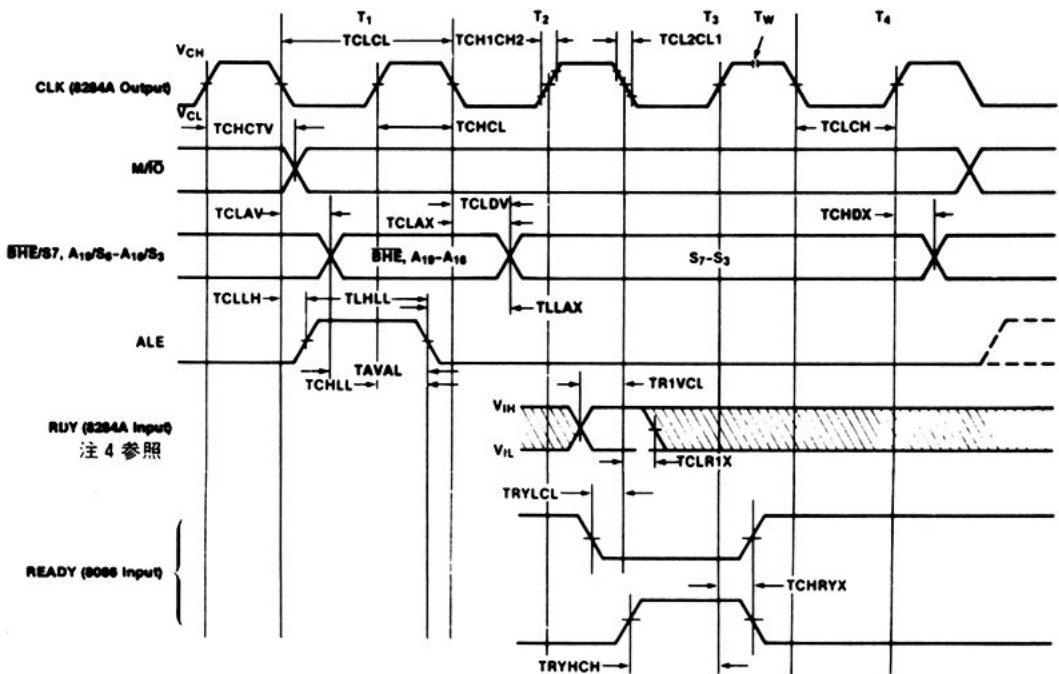
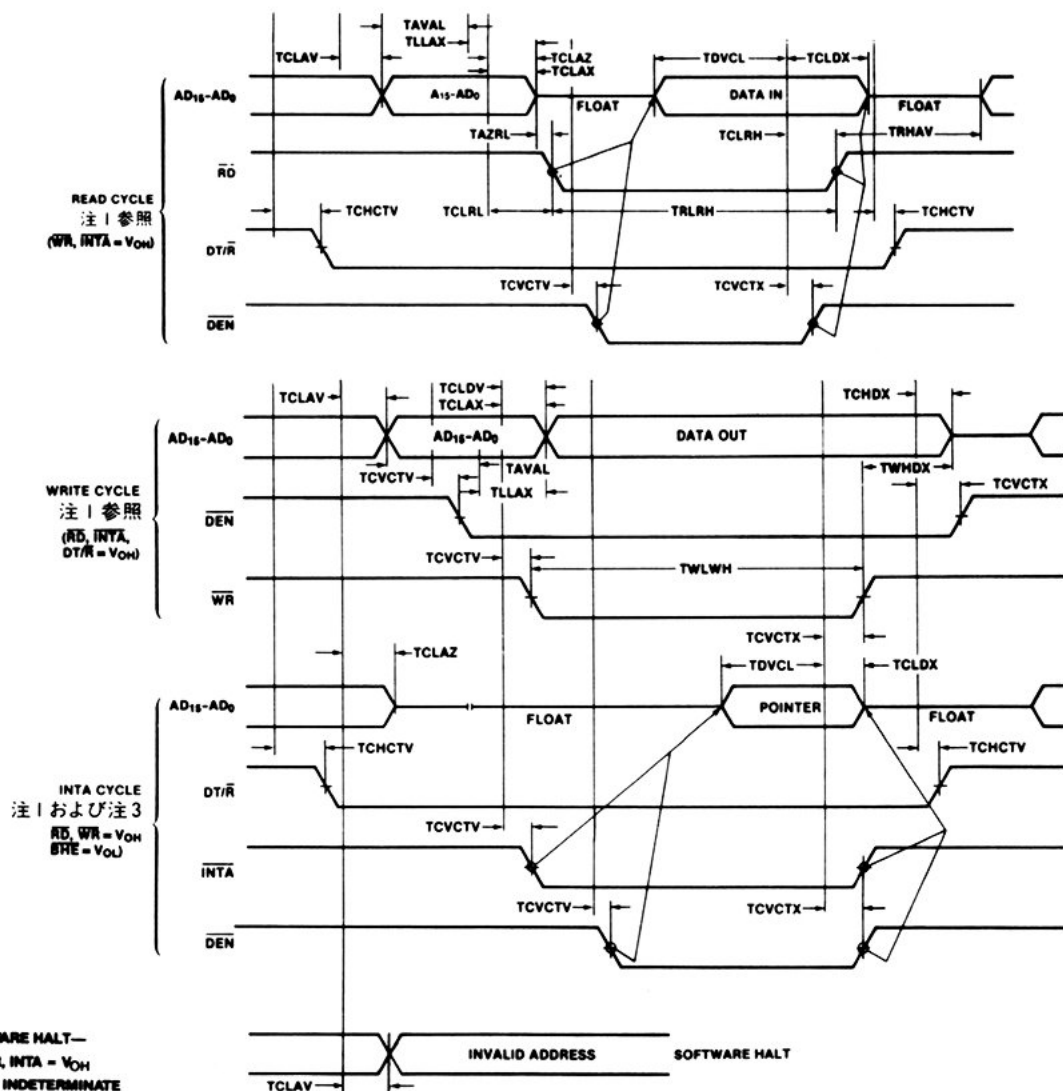
〔注〕 1．参考のために 8284 A または 8288 の信号を示した．

2．次のクロックでの認識を保証するための非同期信号の設定要求．

3．T_3 とウエイト状態のみに適用．

4．T_2 ステートのみに適用（T_3 内へ 8 ns）．

〔5〕 8086 バスタイミング

〈ミニマムモード〉

注 1．特に指定のない場合，すべての信号は V_{OH} と V_{OL} の間の切り換わり．

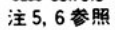
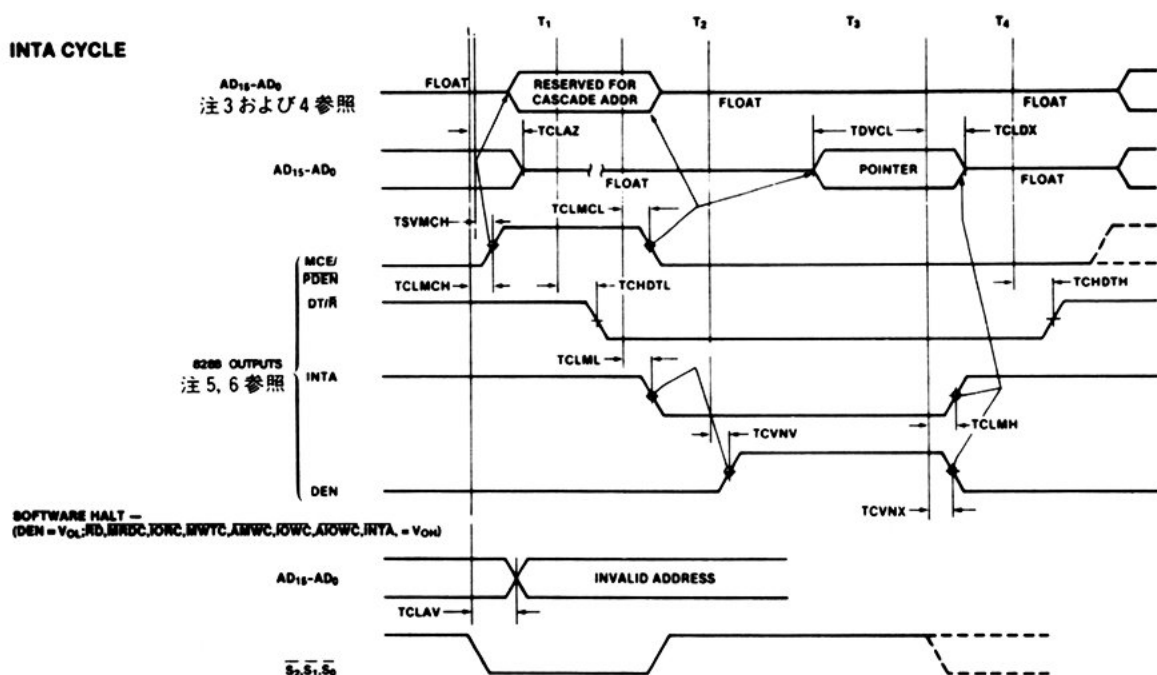
2．RDY は，T_W マシーンステートを挿入するかどうかを決定するため，T_2，T_3 および T_W の終りの近辺でサンプルされる．

3．二つの INTA サイクルは連続して実行される．8086 のローカルアドレス/データバスは INTA サイクルの間フローティングになる．コントロール信号は 2 番目の INTA サイクルのためのものを示す．

4．8284A の信号は参考のためにのみ示してある．

5．特に指定のない場合，すべてのタイミング測定は 1.5V のところで行われている．

〈マキシマムモード〉

T1
T2
T3
T4
TW
CLK
VCH
VCL
TCLCL
TCH1CH2
TCL2CL1
TCLAV
TCHCL
TCLCH
QS0,QS1
TCHSV
TCLSH
(SEE NOTE 8)
TCLDV
TCLAX
TCHDX
S7-S3
TSVLH
TCLLH
TCHLL
ALE (8288 OUTPUT)
注5参照
RDY (8284A INPUT)
TR1VCL
TCLR1X
TRYLCL
READY (8086 INPUT)
TCHRYX
TRYHSH
TRYHCH
READ CYCLE
TCLAZ
TDVCL
TCLDX
A15-AD0
FLOAT
DATA IN
TAZRL
TCLRH
TRHAV
RD
TCHDTL
TCLRL
TRLRH
TCHDTH
DT/R
TCLML
TCLMH
8288 OUTPUTS
注5, 6参照
MRDC OR IORC
TCVNV
DEN
TCVNX
-(see note 8)
WRITE CYCLE
AD15-AD0
DATA
AMWC OR AIOWC
MWTC OR IOWC

注1．特に指定のない場合はすべての信号は V_{OH} と V_{OL} の間の切り換わり．

2．RDY は，T_W マシーンステートを挿入するかどうかを決定するため，T_2，T_3，T_W の終りの近辺でサンプルされる．

3．カスケードアドレスは最初と 2 番目の INTA サイクルの間で有効である．

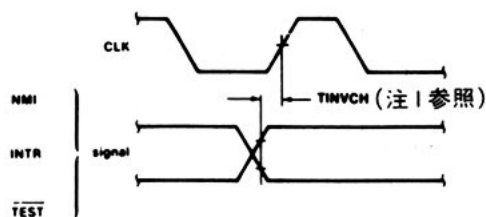
4．二つの INTA サイクルは連続して実行される．8086 のローカルアドレス/データバスは二つの INTA サイクルの間フローティングになる．ポインタアドレスのためのコントロールは 2 番目の INTA サイクルに対して示している．

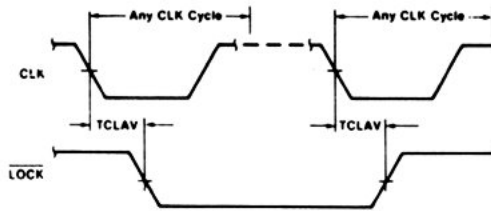
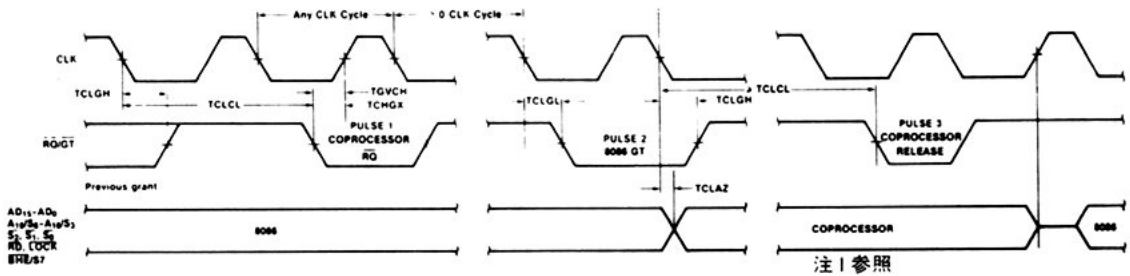
5．8284A または 8288 の信号は，参考のためにのみ示してある．

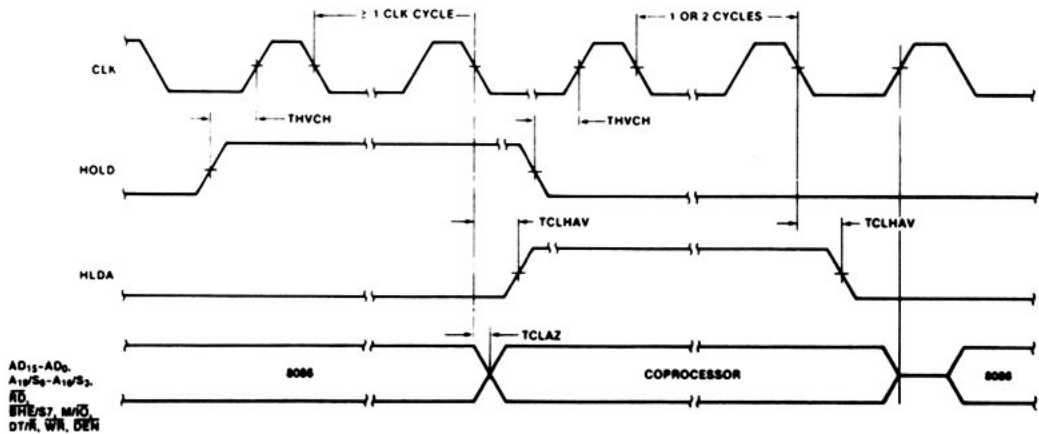
6．8288 コマンドとコントロール信号（$\overline{\text{MRDC}}$，$\overline{\text{MWTC}}$，$\overline{\text{AMWC}}$，$\overline{\text{IORC}}$，$\overline{\text{IOWC}}$，$\overline{\text{AIOWC}}$，$\overline{\text{INTA}}$ および DEN）の発生は，アクティブハイの 8288 CEN の後になる．

7．特に指定のない場合，すべてのタイミング測定は 1.5V のところで行われている．

8．T_4 のすぐ前のステートでのステイタスの不活性．

〔6〕 非同期信号の確認

注1．次のクロックでの認識を保証するためだけの，非同期信号に対する設定要求．

〔7〕 バスロック信号のタイミング(MAXモードのみ)

〔8〕 リクエスト/グランドシーケンスのタイミング図(MAXモードのみ)

注1．コ・プロセッサは，示している領域外では危険な競合なしにバスをドライブすることはできない．

〔9〕 HOLD/HOLDA タイミング(MINモードのみ)

付録6. マルチバスコネクタ信号一覧

	ピン	(部品側)		ピン	(回路側)	
		記号	説明		記号	説明
電源	1	GND	信号 GND	2	GND	信号 GND
	3	+5V	+5 V_{dc}	4	+5V	+5 V_{dc}
	5	+5V	+5 V_{dc}	6	+5V	+5 V_{dc}
	7	+12V	+12 V_{dc}	8	+12V	+12 V_{dc}
	9	−5V	−5 V_{dc}	10	−5V	−5 V_{dc}
	11	GND	信号 GND	12	GND	信号 GND
バスコントロール	13	BCLK/	バスクロック	14	INIT/	初期化
	15	BPRN/	バス優先入力	16	BPRO/	バス優先出力
	17	BUSY/	バスビジイ	18	BREQ/	バス要求
	19	MRDC/	メモリリードコマンド	20	MWTC/	メモリライトコマンド
	21	IORC/	I/O リードコマンド	22	IOWC/	I/O ライトコマンド
	23	XACK/	XFER アクノレージ	24	INH1/	禁止1はRAMを禁止
バスコントロールおよびアドレス	25		予約	26	INH2/	禁止2はPROMまたはROMを禁止
	27	BHEN/	バスハイイネーブル	28	AD10/	アドレスバス
	29	CBRQ/	共通バス要求	30	AD11/	
	31	CCLK/	定常クロック	32	AD12/	
	33	INTA/	割込みアクノレージ	34	AD13/	
割込み	35	INT6/	並列割込み要求	36	INT7/	並列割込み要求
	37	INT4/		38	INT5/	
	39	INT2/		40	INT3/	
	41	INT0/		42	INT1/	
アドレス	43	ADRE/	アドレスバス	44	ADRF/	アドレスバス
	45	ADRC/		46	ADRD/	
	47	ADRA/		48	ADRB/	
	49	ADR8/		50	ADR9/	
	51	ADR6/		52	ADR7/	
	53	ADR4/		54	ADR5/	
	55	ADR2/		56	ADR3/	
	57	ADR0/		58	ADR1/	
データ	59	DATE/	データバス	60	DATF/	データバス
	61	DATC/		62	DATD/	
	63	DATA/		64	DATB/	
	65	DAT8/		66	DAT9/	
	67	DAT6/		68	DAT7/	
	69	DAT4/		70	DAT5/	
	71	DAT2/		72	DAT3/	
	73	DAT0/		74	DAT1/	
電源	75	GND	信号 GND	76	GND	信号 GND
	77		予約	78		予約
	79	−12V	−12 V_{dc}	80	+12V	−12 V_{dc}
	81	+5V	+5 V_{dc}	82	+5V	+5 V_{dc}
	83	+5V	+5 V_{dc}	84	+5V	+5 V_{dc}
	85	GND	信号 GND	86	GND	信号 GND

索　引

ア　行

カ　行

サ　行

<アルファベット索引>

索　　　引

著者略歴

昭和36年　大阪大学工学部通信工学科卒業

現　　在　富士システムリサーチ（株）

図解 16ビットマイクロコンピュータ 8086の使い方

昭和57年10月30日　第1版第1刷発行

昭和58年7月20日　第1版第5刷発行

OHM・OHM・OHM 著者承認 検印省略

著　者　井出裕巳（いでひろみ）

発行者　株式会社 オーム社
代表者　種田則一

発行所　株式会社 オーム社
郵便番号 101
東京都千代田区神田錦町3-1
振替　東京6-20018
電話　03(233)0641(大代)

Printed in Japan

組版　緑新社　印刷　秀好堂　製本　協栄製本

落丁・乱丁本はお取替えいたします

ISBN 4-274-07130-8

書名	著者	判型	定価
だれにも わかる マイコンの考え方・使い方	大條・吹浦共著	A５判	**定価 1700円**
マイコン入門心得帖	平松・森本共著	四六判	**定価 1200円**
続・マイコン入門心得帖	平松・森本共著	四六判	**定価 1200円**
マイコン実験と工作マニュアル	北川一雄著	A５W	**定価 1900円**
続・マイコン実験と工作マニュアル	北川一雄著	A５W	**定価 2300円**
図解 初めてマイコンを学ぶ人のために	大原茂之著	A５判	**定価 1400円**
図解マイコンの基礎知識	矢田光治著	A５判	**定価 2300円**
図解マイコンの使い方	小牧・大條共著	A５判	**定価 1400円**
図解マイクロ コンピュータ Z-80の使い方	横田英一著	A５判	**定価 1700円**
マイクロ コンピュータ MC 6809の考え方	曽和将容著	A５判	**定価 2300円**
図解マイコンのプログラム	平松・守屋・斉藤共著	A５判	**定価 1900円**
図解 マイコンのための BASIC 入門	小牧・大原共著	A５判	**定価 1500円**
図解 マイコンのためのアセンブラ入門	大原・倉田共著	A５判	**定価 1500円**
図解マイコン のための オペレーティングシステム入門	大原・倉田共著	A５判	**定価 1700円**
図解マイコン のための インタフェース入門	大原・北沢共著	A５判	**定価 1700円**
図解 マイコンのインタフェース	平松・斉藤共著	A５判	**定価 1800円**
マイクロコンピュータ入門	国分・藤井共著	A５判	**定価 1800円**
ミニコン・マイコン入門	藤井・桑原共編	A５判	**定価 2000円**
マイクロコンピュータの基礎	矢田光治著	A５判	**定価 1500円**
パーソナルコンピュータ入門	平松啓二監修	A５判	**定価 1600円**
パーソナルコン ピュータのための CP/M 入門	福田・西岡共著	A５判	**定価 2000円**
パーソナルコン ピュータのための 実用BASICプログラミング	安田 永著	B５判	**定価 2200円**
ワンチップマイコン入門	国分・浅見・野村共著	A５判	**定価 1700円**
産業用マイクロコンピュータの基礎と応用	正田・泥堂 中島・松本 共編	A５判	**定価 2300円**
制御用マイコンの作り方・使い方	北川一雄著	B５判	**定価 2900円**
エンジニア のための 絵ときマイクロコンピュータ	吉本久泰著	A５判	**定価 1900円**
コンピュータ安全管理マニュアル	岡本・田口共著	A５判	**定価 2800円**

定価は変更されることがありますのでご了承下さい